VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIAGENTNÍ SIMULAČNÍ MODEL PRO LETECKÉ FORMACE

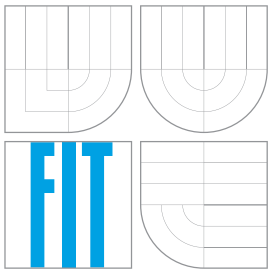
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH ŠALBABA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIAGENTNÍ SIMULAČNÍ MODEL PRO LETECKÉ FORMACE

MULTIAGENT SIMULATION MODEL FOR FLIGHT SQUADRONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH ŠALBABA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FRANTIŠEK ZBOŘIL, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem modelu pro simulaci a vizualizaci taktik leteckých soubojů. Představuje leteckou problematiku a popisuje proces jejího modelování pro simulaci. Popisuje základy leteckých soubojů, jejich principy a význam vybraných taktik. Použitím programovacího jazyka Jason jsou vytvořeni agenti ovládající vybrané taktiky. Tento proces je popisován od rozboru jednotlivých taktik až po konečnou implementaci v programovacím jazyku Jason. Na závěr je ověřen model simulováním leteckých soubojů.

Abstract

This thesis deals with designing a model for simulation and visualisation of air combat tactics. It introduces air combat environment and describes a process for its modeling. Thesis describes basics of air combat, its principles and importance of chosen tactics. Using Jason programming language, artificial agents are created and chosen tactics are implemented. The process of implementing tacitcs is described from analysis to final implementation. Finaly, various types of agents are tested in mock combat against each other. declaration

Klíčová slova

Multiagentní programování, Jason, BDI agenti, základní letecké bojové manévry

Keywords

Multiagent programming, Jason, BDI Agents, Basic air combat maneuvers

Citace

Vojtěch Šalbaba: Multiagentní simulační model pro letecké formace, diplomová práce, Brno, FIT VUT v Brně, 2013

Multiagentní simulační model pro letecké formace

Prohlášení

.....
Vojtěch Šalbaba
22. května 2013

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Františkovi Zbořilovi, Ph.D. za zajímavé téma a odbornou pomoc.

Obsah

Kapitola 1

Úvod

Touha létat je člověku vlastní od pradávna. Od prvních neúspěšných pokusů proběhlo letectví překotným vývojem a brzy našlo uplatnění ve vojenství. Nejdříve v roli průzkumné, pak v roli řízení dělostřelecké palby a bombardovací. Jediným spolehlivým způsobem jak zabránit nepříteli v provádění těchto úkolů bylo vždy prostřednictvím jiného letadla. Tak vznikly za první světové války letadla výhradně určená k ničení letadel nepřítele. Piloti zdokonalovali své schopnosti a vytvářeli první taktiky vedoucí k získání výhody nad nepřítelem. Po překotném vývoji v první světové válce došlo k dalšímu pilování taktik ve světové válce druhé. Některé taktiky vyvinuté během první a druhé světové války zůstaly platné i v dnešní době moderního vojenství, jiné brzy zastaraly technickým vývojem letadel a jejich zbraní.

Pro výcvik pilotů se v dnešní době používají počítačové simulace, ve kterých může pilot získávat zkušenosti bez rizika. Základním předpokladem pro získání opravdu použitelných zkušeností a ne škodlivých zlozvyků je kvalitní protivník, schopný uvěřitelné a realistické reakce na vývoj situace v prostředí. Vytvoření takové umělé inteligence, která bude zároveň schopna reagovat v reálném čase je velmi obtížný a bývá největší slabinou jinak téměř dokonalých simulátorů. Tato práce se zabývá návrhem takového minimálního modelu leteckých soubojů, ve kterém by běžně používané taktiky měly opodstatnění a přinášely účastníkům měřitelnou výhodu nad těmi, které taktiky neovládají.

Umělá inteligence pro tento model je zpracována v agentním programovacím jazyce Jason. V první kapitole jsou uvedeny základy agentního programování a jeho hlavní výhody a nevýhody. Samostatnost agentů a jejich schopnost se samostatně rozhodovat o vykonávaných akcích se výborně hodí pro návrh umělé inteligence pro letecké simulace. V dalších kapitolách jsou probrány základní pojmy z oboru letectví a vysvětleny základní bojové manévry, které jsou pak použity pro implementaci multiagentní simulace leteckých soubojů v poslední kapitole. Součástí poslední kapitoly je ověření, zda agenti ovládající taktiky, které jsou skutečně používány piloty, mají v modelovaném prostředí výhodu nad ostatními.

Tato diplomová práce navazuje na semestrální projekt *Multiagentní simulační model pro letecké formace* řešený a obhájený v roce 2013. V rámci semestrálního projektu byly řešeny první dva body zadání diplomové práce. Z semestrálního projektu je převzata část teoretického úvodu v kapitole 2 a část kapitoly 3.

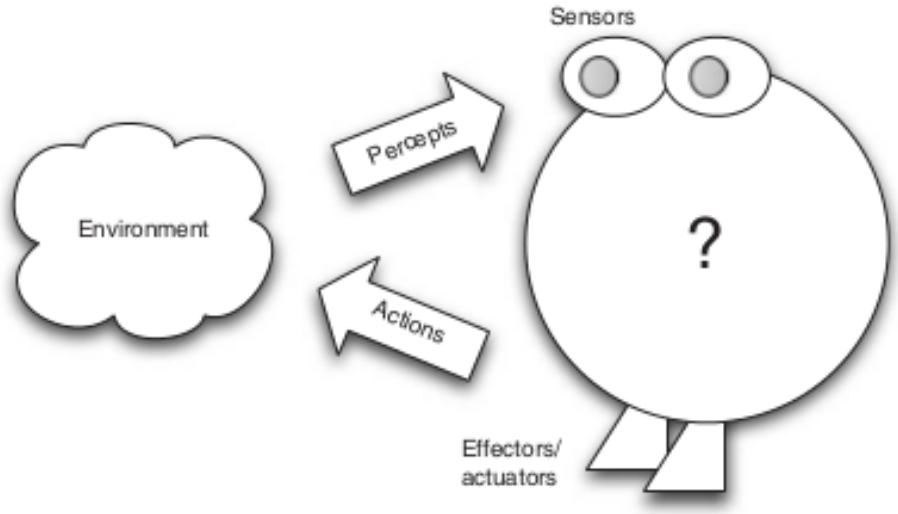
Kapitola 2

Agentně orientované programování

2.1 Co je to agent

> We consider agents to be systems that are situated in some environment. By this, we mean that agents are capable of sensing their environment (via sensors), and have a repertoire of possible actions that they can perform (via effectors or actuators) in order to modify their environment. The key question facing the agent is how to go from sensor input to action output: how to decide what to do based on the information obtained via sensors. [3, str. 1]

Volně přeloženo: Za agenty považujeme takové systémy, které jsou situovány v nějakém prostředí. Tím myslíme, že agenti jsou schopni toto prostředí vnímat (pomocí senzorů) a mají k dispozici akce, které mohou vykonávat (pomocí aktuátorů nebo efektorů) a které mají na prostředí vliv. Klíčová otázka, která stojí před agentem je jak na základě vstupnách vjemů vybrat vhodnou akci: jak se rozhodnout, co dělat na základě senzory získaných informací.



Obrázek 2.1: Jednoduché schéma agenta. Senzory (sensors), efektory (effectors), vjemy (percepts), akce (actions) a prostředí (environment). Zdroj: [3]

Na obrázku 2.1 je znázorněno jednoduché schéma agenta. Agent (na obrázku vpravo) vnímá vjemy z prostředí pomocí senzorů a pomocí efektorů vykonává akce, kterými prostředí

zpětně ovlivňuje.

Agent je stvořen za nějakým účelem, s nějakým cílem. Takový agent, který se snaží tohoto cíle dosáhnout co nejefektivněji s ohledem na své znalosti a dovednosti, se pak nazývá racionální agent. Racionální agent (někdy též nazývaný inteligentní agent) je [3]:

Autonomní Samostatný a nezávisle jedná tak, aby splnil námi stanovené cíle. Agent tedy nemá plně svobodnou vůli jako člověk, je omezen cíli, které jsou mu delegovány. Jeho spektrum voleb, jak těchto cílů dosáhnout, je pak omezeno plány, které jsou mu k dispozici. Plány omezují způsoby, jak může cílů dosáhnout. Jakým způsobem ale seskládá jednoduché plány do komplexních plánů tak, aby dosáhl přidělených cílů, je na agentovi.

Reaktivní Pohotový a adekvátně reaguje na změny prostředí. Ve skutečném světě málokdy jde „vše podle plánu“. Plány se mění. Pokud dojde k tomu, že plán nemůže být splněn, agent musí být schopen zvolit alternativní plán.

Proaktivní Sám se chápe iniciativy a ovlivňuje prostředí v souladu se svými cíli. Pokud je agentovi přidělen nějaký cíl, tak je očekáváno, že agent bude pracovat tak, aby cíle dosáhl. Požadavek proaktivity vylučuje pasivní agenty, kteří se nikdy o nic nepokusí.

Sociální Agent má schopnost koordinovat svou činnost s ostatními agenty tak, aby dosáhl svých cílů. Je užitečné, pokud agenti mají k dispozici pohodlný nástroj pro výměnu svých znalostí, cílů a plánů.

2.1.1 Agentně orientované programování

Co je to agent bylo uvedeno v minulé sekci, k čemu ale agenti jsou? Je důležité si uvědomit, že agentně orientované programování je, stejně jako programování objektové nebo funkcionální, pouze jedním z mnoha přístupů k strukturování programů. Mezi význačné rysy agentně orientovaného programování patří decentralizovaný přístup a autonomnost programových jednotek. Objekt, tak jak jej známe z objektového programování, vystavuje navenenek metody, které pak ostatní objekty volají pomocí zasílání zpráv. Oproti tomu agent si nechává nad sebou plnou kontrolu a jak vyřídí zaslanou zprávu (a hlavně zda ji vyřídí) je plně na něm. Zároveň je běžné aby agent prováděl akce z vlastní iniciativy. Dá se říci, že agenti jsou pokračováním konceptu objektů.

Použití agentů vede k větší decentralizovanosti. Každý agent je samostatnou jednotkou. Jeho provázanost se zbytkem aplikace bývá minimální. Pokud se agentovi zadá nějaký úkol, zodpovědnost na splnění úkolu od té chvíle leží na agentovi. Můžeme se na něj spolehnout, že se bude proaktivně snažit úkol splnit. Není již dále nutné na něj dohlížet a kontrolovat ho.

> Agenti jsou běžně považováni za samostatné entity, protože se dovedou sami starat o množinu svých vnitřních odpovědností. Agenti jsou také interaktivní entity, které jsou schopné zasílat a vyřizovat široké spektrum zpráv. Tyto zprávy mohou způsobit volání zpráv, ale také informovat ostatní agenty o důležitých událostech, ptát se jich na informace nebo být odpovědí na předchozí dotaz. Protože agennti jsou autonomní, mohou komunikaci zahájit dle svého uvážení a na zprávu odpovědět libovolným způsobem. Jinými slovy, o agentech lze uvažovat jako o objektech které dokážou říct nejen „proveď“, ale i „neprovedu“. Přeloženo z Objects and Agents Compared [7].

Padgham a Winikov v [8] na agenty nahlíží jako na snaživé motivované zaměstnance, a na objekty jako na zaměstnance spolehlivé, ale pasivní. Pasivní zaměstnanec (objekt) za námi bude chodit s každým problémem, a to nám jako managerům (programátorům) vytváří nemalou administrativní zátěž. Oproti tomu motivovaný zaměstnanec (agent) je mnohem efektivnější a stačí mu zadat cíl, který on pak samostatně splní. Tím nám uvolňuje ruce pro další práci. Takového zaměstnance pak můžeme nechat pracovat i z domu (decentralizovanost).

2.2 BDI agenti

BDI agenti vycházejí z filozofického modelu lidského chování BDI (Belief, Desire, Intention; Představy, Přání a Záměry), tak jak je uvedl Bratman v [4]. V této práci nebudeme zkoumat hlubší matematické základy BDI logiky, která z této filozofie vychází, ale soustředíme se na její využití při návrhu agentů. V [3] popisují autoři představy, přání a záměry takto:

Představy (beliefs) Informace o prostředí. Tyto informace mohou být neaktuální, nepřesné nebo i lživé.

Přání (desires) Cíle, kterých by mohl agent chtít dosáhnout. To, že agent nějaký cíl má, ještě nemusí znamenat, že bude jednat tak, aby ho dosáhl. Je přirozené, aby agent měl i cíle, které se navzájem vylučují. O přáních je často uvažováno a hovořeno jako o agentových možnostech.

Záměry (intentions) Zvolené způsoby, jak dosáhnout přání. Pro snadnější pochopení je možno si představit, že agent začne s nějakým přiděleným cílem - podívá se na své možnosti jak cíle dosáhnout a z těchto možností si vybere ty, které jsou v aktuální situaci aplikovatelné. Z takto zvolených možností se stávají záměry. Záměry mohou mít podcíle, které vedou k dalším záměrům a tak dále - proces je rekurzivní, dokud nenarazí na atomické, přímo proveditelné záměry - akce.

Padgham a Winikoff pak uvádějí v [8] tyto základní koncepty pro BDI agenty:

Akce Způsoby, jakými může agent ovlivňovat prostředí.

Vjemy Relevantní informace o prostředí.

Události Relevantní informace o změnách v prostředí (vjemy jsou pak podmnožinou událostí).

Cíle Cíle, kterých je třeba dosáhnout (měly by být konzistentní).

Představy Informace o prostředí (neměnné).

Plány Způsoby, jakými dosáhnout cílů.

Zprávy Nezbytná sdělelní pro interakci agentů.

Protokoly Specifikace pro „pravidla“ interakce – většinou spojeno s dosahováním cílů.

Při návrhu agentních systému je pro programátora BDI agent dobře mentálně uchopitelný, a i proto se osvědčil. Při tvorbě agenta nestačí, abysme pracovali s tradiční logikou, i když tuto logiku rozšíříme o temporální a nedeterministické prvky. Důvodem je, že čas nutný

k důkazu tvrzení v tradiční logice je potencionálně neomezený, tedy agent by nemohl naplnit svoji reaktivitu [10] (včasné a pohotové reagování na události).

Rao a Beorgeff tedy v [10] používají tři dynamické struktury, které reprezentují agentovy představy, přání a záměry. Obsah těchto struktur se pak v průběhu životního cyklu agenta mění. Řídící smyčka BDI agenta pak je popsána v pseudokódu ve výpisu kódu 2.1.

```
initialize-state();
repeat
    options := option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-imposible-attitudes();
end repeat
```

Kód 2.1: Řídící smyčka BDI agenta. Zdroj: [10]

Na konci každého cyklu generátor možností (`option-generator`) přečte frontu událostí a vrátí seznam možností. Pak `deliberator` vybere podmnožinu těchto možností a přidá je do struktury záměrů (intentions). Pokud existuje záměr provést atomickou akci, agent ji provede (`execute()`). Následně agent vybere frontu nepřečtených vnějších událostí. Interní události se přidávají ihned, jak k nim dojde. Nakonec interpret vyhodnotí, které záměry byly splněny, a které záměry již splnit nelze, a odebere je z struktur přání a záměrů.

2.3 Multiagentní systémy

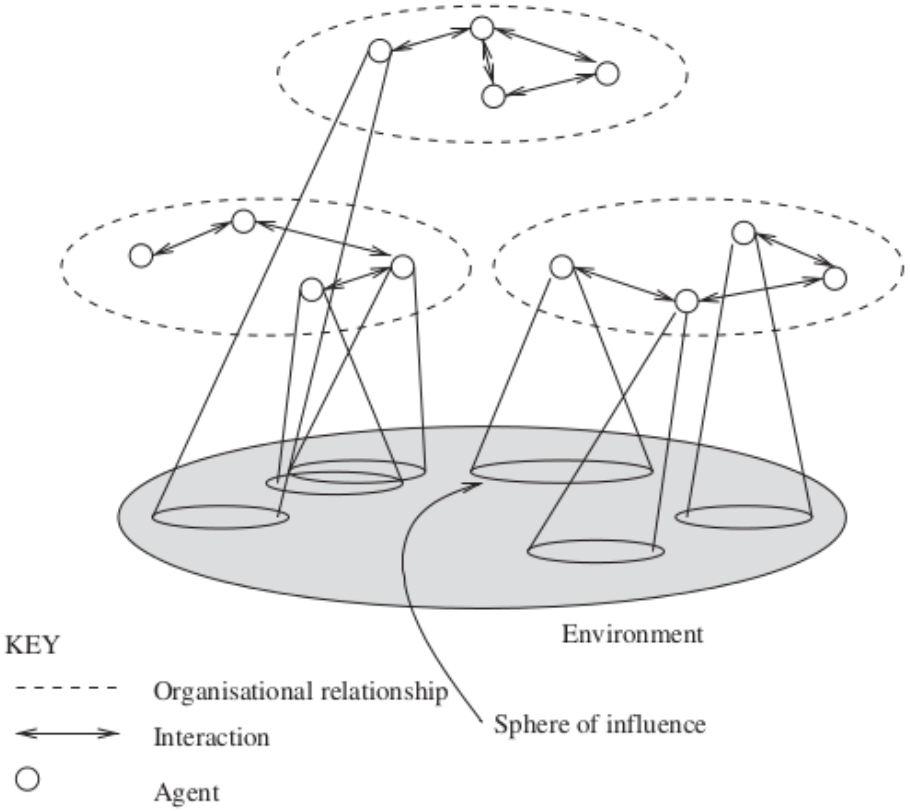
Mulitagentní systémy se zabývají nejen interakcí agenta s prostředím, ale i agentů mezi sebou. Agenti spolu mohou spolupracovat i soupeřit. Není totiž běžné, aby agent byl v prostředí sám. Většinou je jich v jednom prostředí více. Tito agenti mohou mít v tomto prostředí rozdílné možnosti, jak toto prostředí ovlivňovat. Agenti mohou ovlivňovat stejnou část prostředí (sdílet kontrolu nad stejnými prvky), nebo mít vliv na úplně jiné části prostředí. Mezi agenty může být zavedena hiearchie. Agenti nemusí mít vždy úplnou znalost o všech agentech v prostředí. Neznámí konající agenti v prostředí pak činí z pohledu agenta toto prostředí nedeterministickým. Schéma typického multiagentního systému je znázorněno na obrázku 2.2

2.4 Jason

Jason[1] je systém pro realizaci BDI agentů napsaný v programovacím jazyce Java[2]. Základem práce s Jasonem je vytvoření prostředí pro agenta v Javě a následné programování agentů v dialektu agentního programovacího jazyka AgentSpeak [9].

Následující odstace se zabývají jakým způsobem jsou BDI agenti reprezentováni v Jasonu. Základními stavebními jednotkami jsou představy, cíle a plány.

[1] http://jason.sourceforge.net/
[2] http://www.java.com/



Obrázek 2.2: Schéma multiagentního systému. Agenti, prostředí (environment), interakce mezi agenty (interaction), zóny vlivu agentů (sphere of influence), hiearchicky uspořádané skupiny agentů (organisational relationship). Zdroj: [3]

2.4.1 Představy

Nejdřív bude ukázáno, jak jsou v Jasonu reprezenovány představy. Agent má svou bázi představ, která je ve své nejjednodušší podobě sbírkou predikátů, podobně jako v tradičním logickém programování. Například:

```
tall(john).
likes(john, music).
```

Anotace

Významným rozdílem Jasonu oproti Prologu a jiným logickým jazykům jsou anotace. Anotace upřesňují představy. Můžeme o nich smýšlet jako o metainformacích o představách. Některé anotace mají specifikovaný význam, ale programátor může lehce vytvořit nové a význam jim vytvořit tak jak je požadován pro konkrétní aplikaci. Mezi typické anotace patří například jak byla představa získána, nakolik jí lze věřit, dokdy je představa platná. Například:

```
tall(john)[source(amy)].
rains[until(monday)].
```

Anotace sice nijak nerozšiřují vyjadřovací schopnost jazyka (pro dosažení stejného výsledku by stejně tak mohla být přidána nová představa `source(amy, tall(john))`), ale

přináší programátorům tolik potřebné pohodlí. Většina anotací nemá pro interpret Jasonu žádný konkrétní, předdefinovaný význam. Nemůžeme očekávat, že interpret odebere `rains` jen proto, že věří že už je `monday`. Programátor ale může s anotacemi pracovat jak na úrovni agentů samotných, tak na úrovni prostředí, a použít tak anotace k specifickým účelům, jak si je vyžaduje ta která konkrétní aplikace.

Nejdůležitější anotací se speciálním významem je `source` ([3] uvádí, že kvůli `source` anotace vůbec vznikly). Tato anotace je vyplňována přímo interpretem a používá se k zaznamenání, zda představa přišla z senzorů agenta (`source(percept)`), od jiného agenta (`source(agentName)`) nebo se jedná o poznámku agenta samotného („mental note“), pak se označuje `source(self)`.

Negace

V logickém programování se často používá předpoklad uzavřeného světa (closed world assumption). Při uzavřeném světě je předpokládáno, že jakékoli tvrzení, jehož pravdivostní hodnotu agent nezná, je nepravdivé. Dále se často používá negace jako chyba odvození (tvrzení není pravdivé, pokud nepodaří odvodit ze stávajících tvrzení).

Jason oproti tomu používá předpoklad otevřeného světa (open world assumption). Při otevřeném světě je předpokládáno, že pravdivostní hodnota tvrzení je nezávislá na tom, zda ji agent zná. Jason dále používá tzv. silnou negaci označovanou operátorem $\sim$. Tvrzení může mít tedy tři pravdivostní hodnoty:

`rains`**:** Tvrzení je pravdivé

$\sim$`rains`**:** Tvrzení je nepravdivé

jinak: O tvrzení agent nemá žádné informace

Odvozování faktů

Podobně jako v jiných logických jazycích, i v Jasonu jsou nástroje pro odvozování nových faktů z faktů stávajících. Zápis je velmi podobný Prologu a můžeme v něm používat i anotace. Ukázkou můhou být následující příklady:

```
mother(A, B) :- child(B, A) & female(A).
likely_colour(B) :- colour(B)[source(S)] & (S~== self | S~== percept).
```

V prvním případě bylo vytvořeno jednoduché pravidlo pro odvození nového faktu z dvou stávajících. Druhé pravidlo je zajímavější - pravidlo pro možnou barvu (`likely_colour`) odvozujeme, pouze pokud si agent zaznačil barvu sám (`source(self)`) nebo pokud ji právě vidí (`source(percept)`). Tím vyřazujeme případ, kdy nám o barvě řekl jiný agent.

V praxi jsou pravidla v Jasonu užitečná pouze pro jednodušší odvození. Delší nebo hluboce rekurzivní výpočty je vhodnější provést v Javě. Jason je řádově pomalejší než (například) Prolog a při hlubokém vnoření má problémy se správou paměti. Navíc nemá žádnou obdobu prologovských zelených a červených řezů pro řízení výpočtu. Pokud má programátor možnost volby, je praktičtější složitější výpočty přesunout z agenta psaného v Jasonu do prostředí psaného v Javě. Pro toto přenesení složitých výpočtu do Javy Jason ale poskytuje příslušné možnosti.

2.4.2 Cíle

Cíle (v anglické literatuře goals) jsou jedním ze základních konceptů agentů [8] [3]. V Jasonu existují dva druhy cílů: cíle, kterých je třeba dosáhnout (achievement goals) a cíle k ověření (test goals). Cíle, kterých je potřeba dosáhnout (achievment goals) budou v této práci nazývány „praktické cíle“. Cíle, které je třeba ověřit (test goals), budou nazývány „ověřovací cíle“.

Praktické cíle

Při zadání praktického cíle je po agentovi požadováno, aby změnil prostředí tak, aby cíl splnil. Takový cíl je uveden vykřičníkem (`!`). Například `!get(beer)`; `!own(house)`. Je pak na agentovi, aby cíl splnil pomocí vhodných plánů. Při zadání praktického cíle je tedy očekáváno, že agent začne jednat.

Ověřovací cíle

Při zadání ověřovacího cíle je požadováno, aby agent ověřil nějaké tvrzení. Ověřovací cíle jsou uvedeny otazníkem (`?`). Jsou velmi podobné logickému odvozování faktů pomocí pravidel. Rozdílem je, že agent, pokud je mu nějaký fakt neznámý, může podniknout kroky, aby ho zjistil. Porovnejme například:

```
stav_v_bance(A)
?stav_v_bance(A)
```

V prvním případě chceme, aby Jason našel nebo odvodil aktuální stav v bance. Pokud ho nezná nebo neodvodí, selže. V druhém případě, pokud agent stav v bance nezná, může (například) do banky dojít a zeptat se, pokud k tomu má příslušný plán.

2.4.3 Plány

Plán má v Jasonu tři části: spouštěcí událost, kontext (podmínku) a tělo. Spouštěcí událost spolu s kontextem se dohromady nazývají hlava. Událost, kontext a tělo jsou odděleny znaky `:` a `<-` následovně:

```
událost : kontext <- tělo.
```

Událost Události mohou vzniknout buď změnou představ (nový vjem atd.) nebo změnou cílů (nový subcíl atd.). U představ i cílů pak rozlišujeme dvě operace: přidání (`+`) a odebrání (`-`). Když dojde k události, která odpovída hlavě v některém z plánů (a kontext je platný), říkáme, že plán je relevantní.

Kontext V agentně orientovaném programování odkládáme rozhodnutí *jak* cíl splnit na co nejpozdější dobu. Kontext plánu udává, kdy je plán aplikovatelný. Interpret pak vybere z aplikovatelných plánů ten, dle kterého bude agent jednat.

Tělo Tělo plánu je posloupnost příkazů, které mají splnit příslušný cíl. Ne každý z příkazů musí být přímo vykonavatelný, v těle lze i přidávat nové subcíle.

Příklad plánů:

```
+!drink : has_drinks(0) <- !get_drink; !drink.
+!drink: gulp.
+!get_drink <- ?has_drinks(X); -+has_drinks(X+1).
```

V prvním případě je v hlavě vyplněný i kontext. V druhém případě kontext chybí. Pokud má plán prázdný kontext, je plán vždy aplikovatelný. V posledním případě je vidět práce s představami.

Událost: Přidání praktického cíle napij se - `drink`.

Kontext: Agent věří, že nemá co pít. Plán je aplikovatelný, pokud dojde k přidání cíle `drink` a agent věří, že `has_drinks(0)`.

Tělo: získej něco k pití a pokus se znova napít

2.4.4 Interní akce

V konečném důsledku všeho plánování je požadováno, aby agent nějak změnil prostředí. K provádění akcí měnících prostředí slouží interní akce. Jako interní akce mohou být také definovány funkce hůře uskutečnitelné v Jasonu. Jason přichází s malým repertoárem předdefinovaných akcí, které jsou především zaměřené na komunikaci agentů a práci se seznamy. Pro aplikačně specifické akce je třeba rozšířit prostředí.

2.4.5 Komunikace mezi agenty

Jason poskytuje velmi dobré nástroje pro komunikaci mezi agenty. Poslání zprávy je pro programátora bezpracné a o doručení se stará Jason. Zprávu je možno poslat jednomu nebo více příjemců. Výchozí prostředí nijak neomezuje kdo komu může posílat zprávy. Jednotliví agenti pak mohou obsahovat další logiku pro vyřazení nevyžádaných zpráv. Zprávu agent odešle interní akcí `.send(Příjemce, sloveso, obsah)`. Každý z příjemců pak obdrží zprávu ve formátu `<Odesilatel, Sloveso, Obsah>`.

Odesilatel Agent, který zprávu odeslal.

Sloveso Jedno ze sloves tell, untell, achieve, unachieve, askOne, askAll, tellHow, untellHow, askHow.

Obsah Obsah zprávy. Jeho význam se liší dle použitého slovesa.

Význam a použití sloves je následující:

Sdělování informací

tell Obsah je interpretován jako představa. Příjemce obdrží událost +obsah[source(odesilatel))]. Příjemce může tuto událost vyhodnotit dle libosti, pokud nespecifikuje jinak, je představa prostě přidána do příjemcovy množiny představ. Příklad:

```
.send(receiver, tell, door(closed))
```

untell Obsah je interpretován jako představa a příjemce obdrží událost -obsah[source(odesilatel)]. Pokud příjemce nespecifikuje jinak, přidá se představa do množiny představ příjemce. Příklad:

```
.send(receiver, untell, door(open))
```

Delegace cílů

achieve Obsah je interpretován jako cíl. Příjemce obdrží událost +!obsah. Pokud příjemce má relevantní plán pro tuto událost, obslouží ji. Příklad:

```
.send(receiver, achieve, open(door))
```

unachieve Obsah je interpretován jako cíl. Příjemce neobdrží žádnou událost, ale pokud měl ke splnění naplánovaný nějaký cíl odpovídající obsahu zprávy, je mu tento plán odebrán ze seznamu cílů ke splnění. Příklad:

```
.send(receiver, unachieve, open(door))
```

Hledání informací

askOne Obsah je interpretován jako představa ve které jsou proměnné. Odesilatel se ptá, zda příjemce dovede naplnit proměnné tak, aby představa dle příjemce byla pravdivá.

askAll Obsah je interpretován jako představa ve které jsou proměnné. Odesilatel se ptá, na všechny možné řešení proměnných aby dle příjemce byla představa pravdivá.

Sdělování plánů

tellHow Odesilatel pošle příjemci plán.

untellHow Odesilatel, žádá aby si příjemce smazal plán ze svého seznamu plánů.

askHow Odesilatel se ptá příjemce na relevantní plány pro určitou událost.

Toto jsou nejběžněji používaná slovesa při komunikaci agentů. Pro podrobné vysvětlení všech sloves odkazuji na [3, str. 119].

2.5 Tvorba prostředí pro Jason agenty

Pro reálné aplikace nestačí výchozí prostředí. Je třeba agentům předávat vjemy a umožnit jim prostředí smysluplně měnit. Toho lze dosáhnout rozšířením výchozího prostředí o nové interní akce a o aplikačně specifický generátor vjemů.

Nové prostředí pro agenty se tvoří jako potomek Java třídy `Environment`. Tato třída pak může implementovat metodu `getPercepts` pro získání vjemů z prostředí. Metoda `getPercepts` je zodpovědná za to, že agent dostane pouze vjemy, které mu náleží. Výchozí implementace metody `getPercept` je ale velmi dobrá a její úprava není běžně nutná.

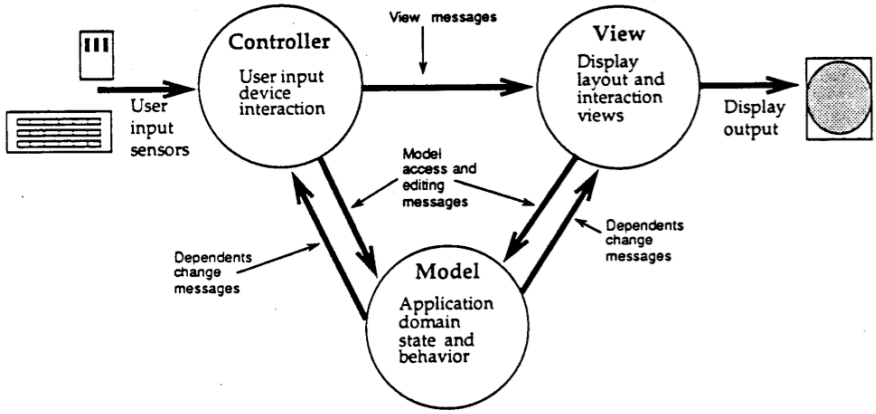
Pro správu vjemů dává rodičovská třída `Environment` k dispozici metody `addPercept`, `removePercept` a `clearPercepts`. Účinek agentových akcí na simulované prostředí je určen definicí metody `executeAction`. Tato metoda má za vstupy jméno agenta a vykonávanou akci a je na ní, aby příslušně změnila prostředí. Jason je ve srovnání s ostatními agentními systémy závislý na volání specifických metod agenty [2].

Při tvorbě prostředí je doporučeno ([3], příklady dodávané spolu s distribucí Jasonu) tvořit prostředí architekturou Model-View-Controller (MVC).

Architektura MVC

> Programování stylem Model-View-Controller (MVC) je aplikací rozdělení projektu na 3 části, kdy se objekty různých tříd starají o problematiku aplikační domény (model), zobrazení (view, pohled) a uživatelovy interakce s modelem a pohledem (controller, řadič). Přeloženo z [5].

Schéma MVC architektury je znázorněno na obrázku 2.3. Architektura MVC je pro tvorbu prostředí velmi vhodná. Agent vytváří vstupy, které zpracovává řadič, který změny promítá do modelu. Uživateli je stav modelu zobrazen pomocí pohledu. Oddělení aplikační domény do modelu programátorovi agentního systému umožňuje následné jednoduché vyměnění simulovaného prostředí za reálné.



Obrázek 2.3: MVC architektura. Řadič (controller) přijímá vstupy ze senzorů (user input sensors), model spravuje data a chování a pohled (view) zodpovídá za výstupy. Zdroj: [5]

Kapitola 3

Taktika stíhacích letadel

Souboje stíhacích letadel jsou velmi komplikovanou záležitostí. V této kapitole bude představena letecká problematika, budou vysvětleny základy stíhacích soubojů ve dvojic i i v týmu. Nejdříve ale budou definovány základní pojmy a ukázány základní bojové manévry.

3.1 Pojmy

Pro pochopení následujících sekcí je třeba nejdříve (alespoň zjednodušeně) vysvětlit pojmy, které budeme používat. Pro lepší představu nahlédněte na obrázek 3.1.

Letový kurz Zeměpisný kurz, kterým letoun letí. Ve stupních, 0-360. Jako 0 (potažmo 360) je určen sever.

Vzájemný kurz V literatuře Angle off Tail (AOT). Úhel, který svírají kurzy dvou letadel. Stíhači letící stejným kurzem mají vzájemný kurz 0. Stíhači letící proti sobě mají AOT 180. Vzájemný kurz je nezávislý na aktuální pozici stíhačů, pouze na jejich orientaci. Vzájemný kurz rozlišujeme kladný a záporný.

Aspekt Úhel, pod kterým pilot jednoho letadla může vidět druhé. Nezávislé na vzájemném kurzu. Může být vyjádřen buď ve stupních, slovně (vlevo vzadu, vpředu) nebo jako hodina na imaginárním ciferníku kde 12 je vpředu a 6 je vzadu (například „na 2 hodinách“, „na 9 hodinách“).

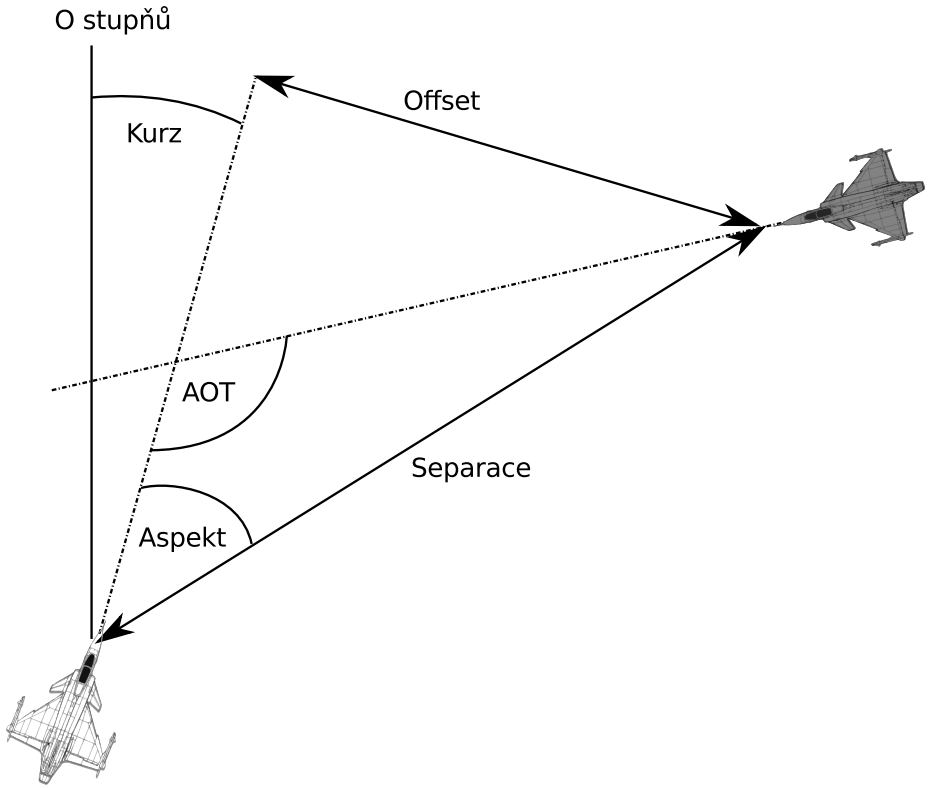
Separace, vzdálenost Vzdálenost mezi dvěma letadly, měřeno „Vzdušnou čarou“

Ofset Vzdálenost letadla od osy druhého letadla

Poloměr zatáčky Poloměr kružnice, kterou stíhač opíše při jeho aktuální rychlosti. Poloměr zatáčky je v přímé úměře k rychlosti stíhače.

3.2 Zbraňové systémy

Stíhací letoun existuje pro zničení ostatních letadel. Letadlo samotné může být považováno pouze za prostředek k vytvořený k umístění zbraňových systémů do pozice pro střelbu. Zbraně stíhačů prošly v průběhu let velkou proměnou,



Obrázek 3.1: Znázornění ofsetu, aspektu, letového kurzu, vzájemného kurzu a separace

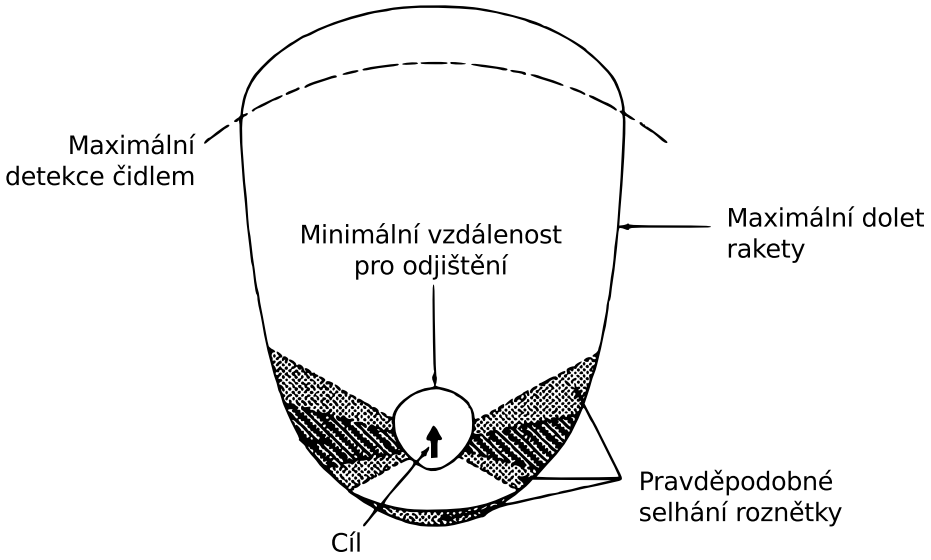
> a každá zbraň měla a má rozdílné požadavky pro její úspěšné použití. Požadavky mohou obsahovat například účinný dostřel, míření, vzájemná pozice stíhače a jeho cíle nebo mnoho jiných faktorů. Všechny požadavky používané zbraně musí být naplněny současně pokud má být zbraň účinně použita. Plnění těchto požadavků a zároveň znemožnění téhož soupeři musí tedy být cílem všech taktik a manévrů.
> Přeloženo z [11, str. 1]

Cílem všech manévrů je dostat se do pozice, umožňující sestřelení protivníka a zabránit v tomtéž soupeři. Tato pozice je závislá na použitých zbraňových systémech - je třeba dát ideální předpoklady pro práci svým zbraňovým systémům a znemožnit práci zbraňových systémů nepřítele. Proto je třeba uvést aspoň základní fakta o zbraňových systémech stíhacích letadel.

Základem výzbroje moderních stíhačů jsou naváděné raketové systémy. Každý zbraňový systém se skládá z jednotlivých komponent. Každá komponenta má určité limity ve kterých pracuje ideálně - mimo tyto limity funguje se sníženou účinností nebo vůbec. Snížením efektivity nebo úplným vyřazením libovolné komponenty je vyřazen celý systém [11, str. 31]. Stručně proberme tyto komponenty a jejich běžné limity.

Pohonná jednotka Motor rakety. Udává maximální dostřel a maximální rychlost rakety. Raketa mimo dostřel není nebezpečná - a stíhač může kličkováním způsobit, že raketa vyčerpá své palivo dříve, než stíhače dostihne. Stíhač střílející na cíl ze zadní polosféry musí být cíli v okamžiku vypuštění rakety mnohem blíže než stíhač střílející z přední polosféry.

Zatímco při útoku zepředu se cíl raketě přibližuje, při útoku zezadu se cíl vzdaluje. Grafické znázornění dostřelu při různých vzájemných pozicích střelce a cíle jsou na obrázku 3.2.



Obrázek 3.2: Znázornění dostřelu při různých vzájemných pozicích střelce a cíle. Upraveno z [11].

Ovládací jednotka Umožňuje raketě manévrovat, ať už ovládacími ploškami, natáčením trysky motoru nebo kombinací. Donucením rakety do manévru, který raketa nemůže následovat stíhač raketu zneškodní.

Roznětka a bojová hlavice Roznětka má za úkol odpálit bojovou hlavici ve vhodný okamžik a ta zničit nepřítele. Pokud roznětka dá pokyn k odpálení ve špatný okamžik, je raketa neškodná. Například roznětka určená k útoku ze zadní polosféry nemusí být vhodná pro útok z polosféry přední nebo z boku. Letadlo vyrobené z netradičních materiálů nemusí být některými typy rozbušek detekováno, atd. Roznětka taktéž potřebuje po odpalu krátkou dobu pro inicializaci. Tím vzniká jistá minimální vzdálenost, pod kterou není raketa účinná.

Senzor Senzor detekuje cíl. Běžně používané jsou senzory tepelné (detekují horké plyny vycházející z trysek letadla), laserové (detekují odraz laserového paprsku od cíle), radarové a podobně. Pokud je cíl schopen snížit detekovatelnost senzorem, střela může selhat. Například tepelné senzory jsou velmi účinné pokud se „dívají“ do trysek nepřátelského stíhače (velký teplotní kontrast) ale pokud nepřítel letí přímo proti raketě, kontrast je mnohem nižší. Radary zase mají problém rozpoznat nízko letící cíle při pohledu shora, kdy odraz od země vytváří velmi silný šum, ve kterém se radarový kontakt ztrácí. Laserový paprsek bývá většinou vyzařován z externího zdroje, typicky letadla, které raketu vystřelilo. Charakteristikou senzoru je také maximální aspekt, ve kterém je ještě schopen detekovat cíl.

Řídící jednotka Vyhodnocuje informace ze senzoru a dává pokyny ovládací jednotce. Pokud cíl ví, jakým způsobem řídící jednotka funguje, může ji donutit udělat špatná rozhodnutí. Například může donutit raketu zmást tak, že k němu poletí po nevhodné křivce a vyčerpá palivo dříve, než se k němu dostane

Shrnutí

Některé komponenty nemusí být přímo v těle rakety, mohou být umístěny například v stíhači, který raketu vystřelil. Výhodou je, že složitá elektronika vyžadovaná pro detekci cíle nezabírá místo v raketě a může být mnohem objemnější a tedy i sofistikovanější. Nevýhoda je zřejmá - při zničení stíhače je zneškodněna i samotná raketa. Běžně používané rakety krátkého dosahu jsou tepelně naváděné a veškeré komponenty jsou obsaženy v těle rakety. Tím může útočník po vystřelení rakety dál manévrovat a bránit se.

Zásah naváděnou raketou není zaručen a cíl rozhodně není bezmocnou pasivní obětí. Pro úspěšnou obranu je důležité znát co nejpodrobněji typ vystřelené rakety a systémy, které používá. Tomu v letadle pomáhá mnoho automatizovaných systémů, ale zkušenost, výcvik a téměř encyklopedické znalosti pilotů o naváděných raketách používaných nepřítelem hrají hlavní roli.

3.3 Základní bojové manévry jednotlivců

Základní bojové manévry jsou stavebními bloky, ze kterých se skládá souboj. Je možné je rozdělit na obranné a útočné.

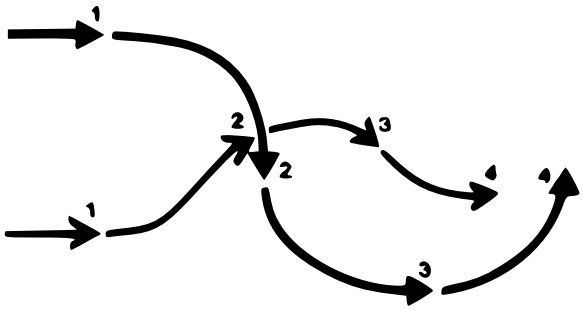
3.3.1 Obranné manévry

Nejlepší obranou proti útočníkovi je samozřejmě nenechat na sebe střílet, a pokud není možno v střelbě útočníkovi zabránit, je třeba zkrátit čas, po který útočník střílet může a učinit mu střelbu co nejobtížnější. Nejobtížnější střelba je na cíle s rychle se měnící vzdáleností a aspektem.

Prodloužení Prodloužení použije obránce v případě, kdy má proti útočníkovi významnou rychlostní převahu a dostatečnou separaci. Srovná se tak, aby měl útočníka na 6té hodině (přímo za sebou) a zvýší separaci. Pokud by obránce neměl dostatečnou počáteční separaci, vystavil by se útočníkovi jako ideální cíl. Po zvýšení separace se obránce může v bezpečí rozhodnout, zda se otočí a na útočníka sám zaútočí, nebo opustí prostor.

Úhyb Letadlo se snaží prudkým zatočením v rovině (většinou směrem k útočníkovi) zvýšit rychlost přibližování a zvýšit rychlost změny aspektu. Opakovaným prováděním úhybu přechází souboj v nůžky.

Horizontální nůžky Letadlo se snaží prudkými změnami směru v horizontální rovině zpomalit svůj dopředný pohyb a „vytlačit“ nepřítele před sebe. Používá se především pokud útočník má stejnou nebo vyšší rychlost než obránce. Díky určitému zpoždění mezi tím, kdy obránce začne úhyb, a tím, kdy útočník úhyb zpozoruje a zareaguje na něj, se útočník dostane velmi rychle mimo obráncův rytmus a dostává jen velmi krátké a obtížné možnosti ke střelbě. Pokud se útočník neodpoutá nebo nemá výrazně obratnější letadlo, obránce ho postupně vytlačí před sebe. Jedná se o velmi agresivní manévr. Krátké horizontální nůžky jsou znázorněny na obrázku 3.3. Útočník začíná ve srovnatelné pozici s obráncem, ale již od časového okamžiku 3 je obránce ve výhodě, a přechází do útoku. Pozice útočníka a obránce se vyměnily.

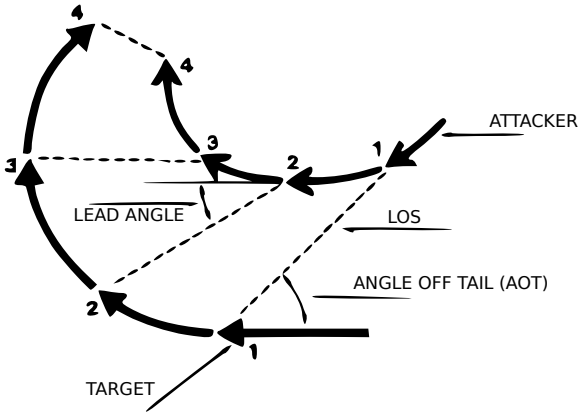


Obrázek 3.3: Horizontální nůžky. Útočník (nahoře) je rychlejší, a pomalejší obránce (dole) může udělat zatáčku o menším poloměru. Po třetím úhybu (časový okamžik 4) je obránce za útočníkem a útočník musí přejít do obrany. Zdroj: [11]

3.3.2 Útočné manévry

Při útoku se snaží útočník dostat obránce před sebe tak, aby byl schopen srovnat jakýkoli obráncův pokus o změnu aspektu a vzájemného kurzu. Útočník při útoku volí jednu ze tří křivek přístupu.

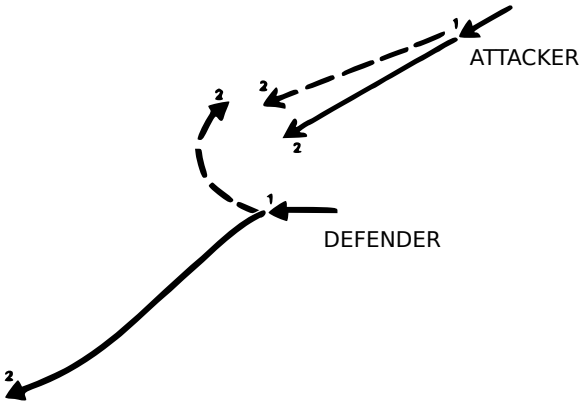
Předsazená (lead) křivka Při „lead“ pronásledování letí útočník na místo před soupeře. „Lead“ se používá pro zkrácení vzdálenosti. Obránce letí po delší křivce než útočník. Vzdálenost mezi nimi se rychle zkracuje. Předsazené pronásledování je znázorněno na obrázku 3.4. Proti útočníkovi používajícímu předsazenou křivku jsou dvě standardní obrany, dle rozdílu v rychlostech mezi útočníkem a obráncem, viz obrázek 3.5.



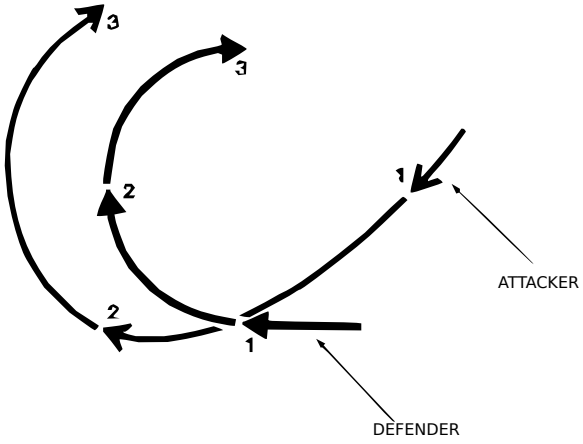
Obrázek 3.4: Předsazená (lead) křivka. Útočník (attacker) rychle zkracuje vzdálenost k nepříteli (target), i když ten letí rychleji. Zdroj: [11]

Přímá křivka Přímá křivka je hlavním metodou pro dosažení pozice za soupeřem. K pronásledování po přímé křivce dochází, když se útočník neustále otáčí tak, aby nos jeho letadla směřoval přímo na nepřítele. [1, kap. 103]

Zpožděná křivka Při „lag“ pronásledování směřuje nos útočníka za ocas soupeře. Používá se pro vylepšení pozice, zachování nebo zvětšení vzdálenosti. Obránce letí po kratší trase. Pronásledování po zpožděné křivce je znázorněno na obrázku 3.6.



Obrázek 3.5: Obrana proti útočníkovi (attacker) používajícímu předsazenou křivku. Pokud se útočník přibližuje rychle, obránce (defender) provede úhyb směrem k útočníkovi (přerušovaně). Pokud je obránce výrazně rychlejší, přejde obránce do prodloužení. Zdroj: [11]



Obrázek 3.6: Zpožděná (lag) křivka. Útočník (attacker) si zachovává pozici za nepřítelem (defender) i přes svou vyšší rychlost. Zdroj: [11]

3.3.3 Přiblížení

Nepřátelská letadla si při setkání často letí vstříc, takzvaně „čelo na čelo“. Toto je pro stíhače, který má jinak na své straně více výhod (lépe manévrující stroj, rychlejší stroj, lepší detekční techniku, lépe vycvičený pilot, domácí prostředí ...) velmi nežádoucí situace. Oba stíhači mají v tomto případě totiž přibližně stejné šance na úspěch - kdo zvítězí se stává záležitostí štěstí. Navíc obránce může po minutí útočníka pokračovat v původním kurzu, je pravděpodobné, že útočník se nestihne otočit pro opakování útoku. Proto je pro útočícího stíhače výhodnější přiblížit se k cíli takovým způsobem, který mu dá útočníkovi výhodnou výchozí pozici.

Ofsetové přiblížení Stíhač získá významný ofset a cíl míjí. Jakmile aspekt nabude cca 80 stupňů, zatočí k němu. Obránci tím dává tři možnosti:

Provést úhyb k útočníkovi Pokud provede úhyb, útočník si tím vynutil souboj, ve kterém má výhodu. Obránce úhybem ztratí energii a vstupuje do něj z horší pozice.

Provést prodloužení od útočníka Pokud provede prodloužení, obránce riskuje, že ho útočník stihne doletět. Pokud nemá výhodu rychlosti aby se dostal mimo dostřel, vystavuje se útočníkovi v ideální pozici pro jeho zbraňové systémy. Zároveň tím obránce opouští svůj původní cíl, ke kterému letěl.

Pokračovat v původním kurzu Pokud bude pokračovat v původním kurzu, je situace podobná jako u prodloužení. Obránce neopouští svůj cíl, doufá, že útočníkovi uletí.

Pro útočníka není ofsetové přiblížení bez rizika a to především pokud je za jeho cílem další, nedetekovaný nepřítel. Pak se při snaze dostat se do výhodné pozice pro střelbu na cíl sám vystaví útoku - pokud se bude bránit, dostane se mezi jeho původní cíl (který může otočit a sám se stát útočníkem) a již zmíněného nedetekovaného nepřítele. Ofsetové přiblížení se kvůli tomuto vážnému nebezpečí téměř nepoužívá nad nepřátelským územím.

Dalším nebezpečím je špatné načasování zatáčky do útoku. Pokud zatočí útočník příliš brzo, obránce mine a souboj přejde v nůžky (kde může mít obratnější obránce navrch), pokud zatočí příliš pozdě, obránce mu může uniknout. Načasování zatáčky je kritické. Příručka pro piloty námořnictva USA [1] uvádí, že zatáčka by měla být načasována tak, aby po celou dobu útočník směroval nosem na svůj cíl. Tím vystavuje obránci nejmenší možný profil a velmi snižuje

Taktéž pokud obránce detekuje útočníkův pokus o ofsetový útok včas, může mu prostě uletět a útočník obránce nestihne dostihnout.

Rozlišujeme levou a pravou ofsetovou trať. Levá trať je ta, kdy se stíhači míjí levou stranou, jak je popsáno na obrázku 3.7.

Ofsetové přiblížení je znázorněno na obrázku 3.8. Plán ofsetového přiblížení, tak jak byl předán zadavatelem je na obrázku 3.7.

3.4 Taktiky bojových formací

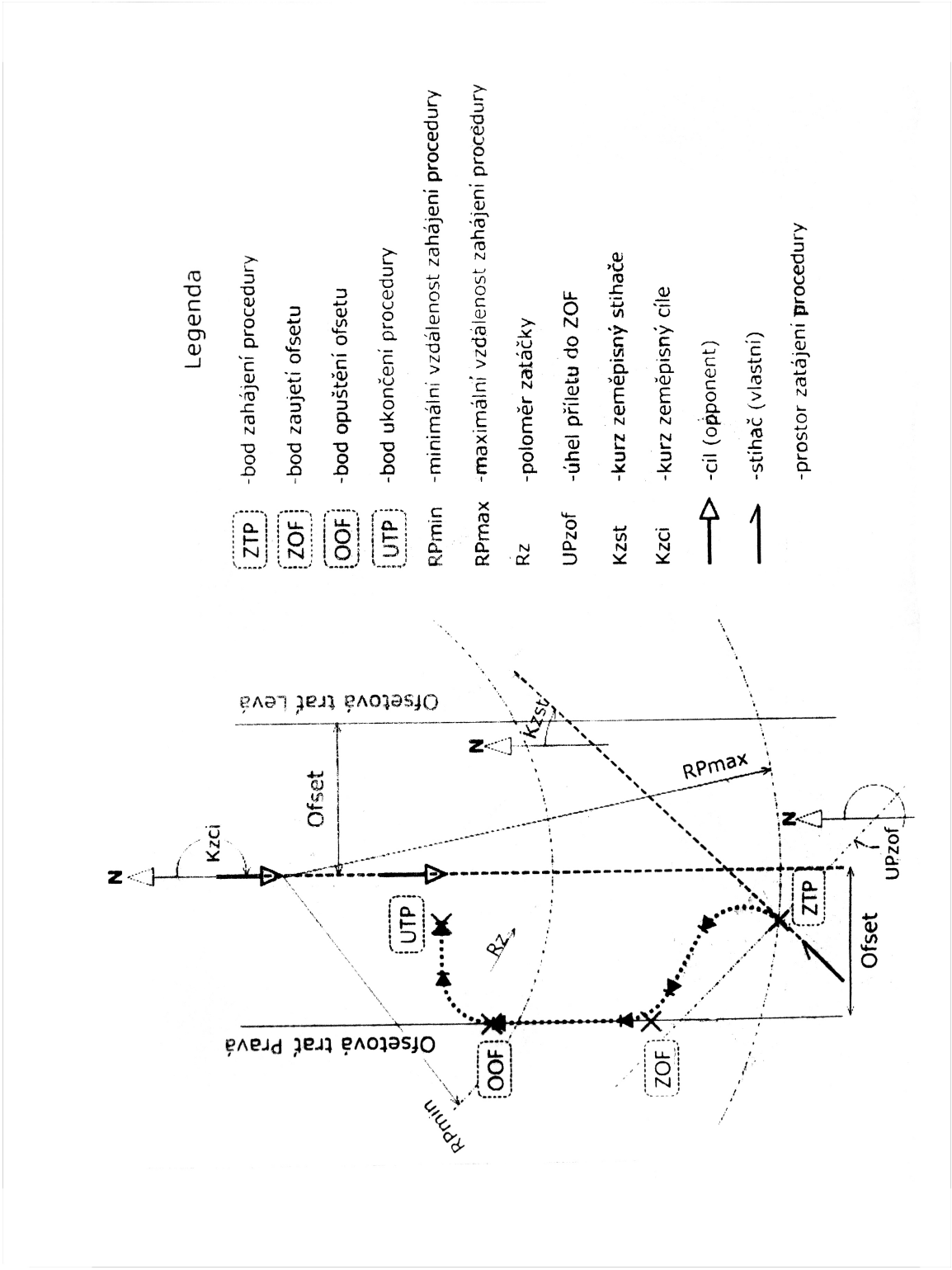
Nyní se podívejme na základní způsoby, jakými mohou spolupracovat stíhači v rámci jedné formace. Pro přehlednost budeme označovat stíhače jedné strany vždy barvou - tedy modrý stíhač spolupracuje s ostatními modrými stíhači a jeho nepřítelem jsou stíhači červení. Většina manévrů spoléhá na efekt takzvaných „rudých očí“, kdy se červený stíhač soustředí na jednoho modrého stíhače, zatímco druhý modrý stíhač vmanévruje do ideální pozice pro střelbu.

3.4.1 Drag and Bag

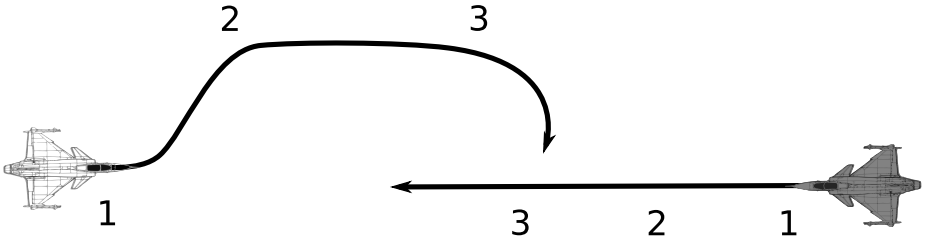
Ohrožený modrý stíhač A se dobrovolně vystaví útoku stíhače červeného. Je praktičtější pokud modrý stíhač A dá červenému stíhači pouze ideu, že by útok mohl provést a sám nebude v reálném ohrožení. Typicky se drží těsně mimo dostřel. Druhý modrý stíhač pak manévruje tak aby červeného stíhače sestřelil dříve než červený stihne sestřelit ohroženého stíhače modrého. Výhoda je na straně modrých - prudký manévr modrého stíhače A je vždy následován výrazně menším manévrem stíhače červeného - tedy modrý stíhač B má jednoduší střelbu než červený stíhač na modrého stíhače A. Taktika Drag and Bag je znázorněna na obrázku 3.9

3.4.2 Ofsetové přiblížení formace

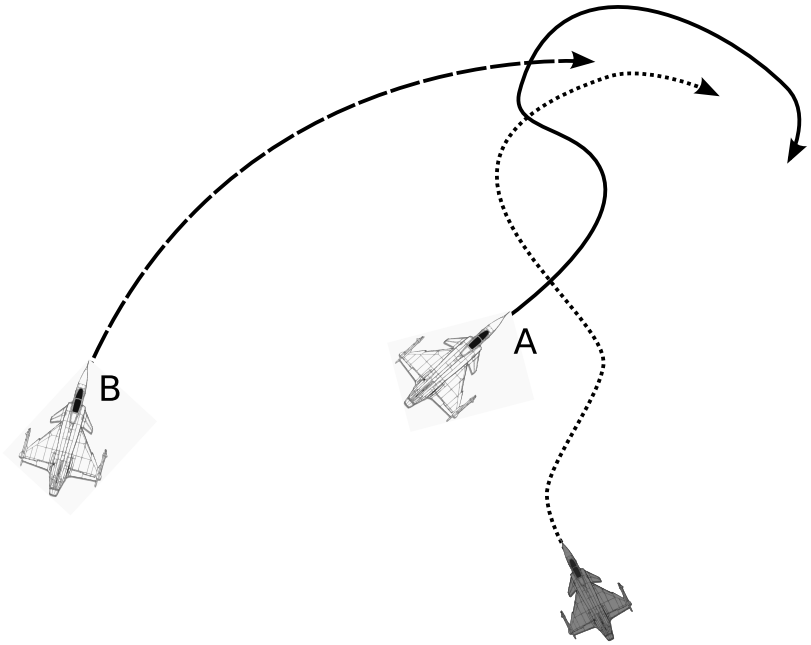
Podobně jako ofsetové přiblížení jednotlivce může dvojice stíhačů provést ofsetové přiblížení ve dvojici. Pokud oba zvolí stejnou stranu, zvýší tím velmi svoji bezpečnost. Stíhači mezi



Obrázek 3.7: Ofsetové přiblížení tak jak bylo předáno zadavatelem



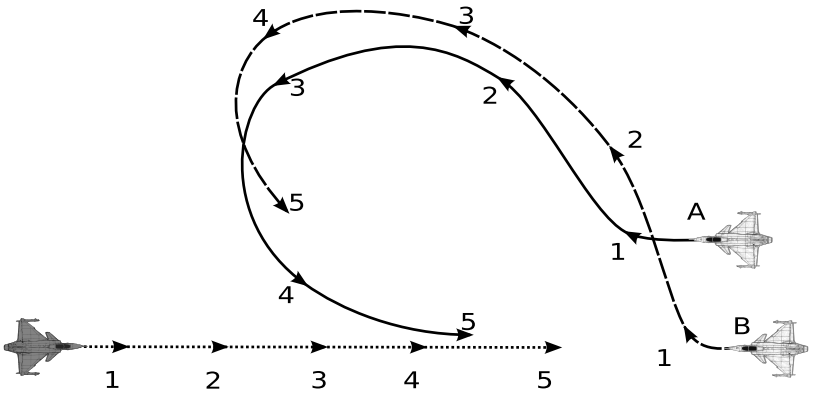
Obrázek 3.8: Ofsetové přiblížení. Útočník (vlevo) získá ofset (ofset získán v čase 2) a při míjení protivníka k němu zatočí (čas 3).



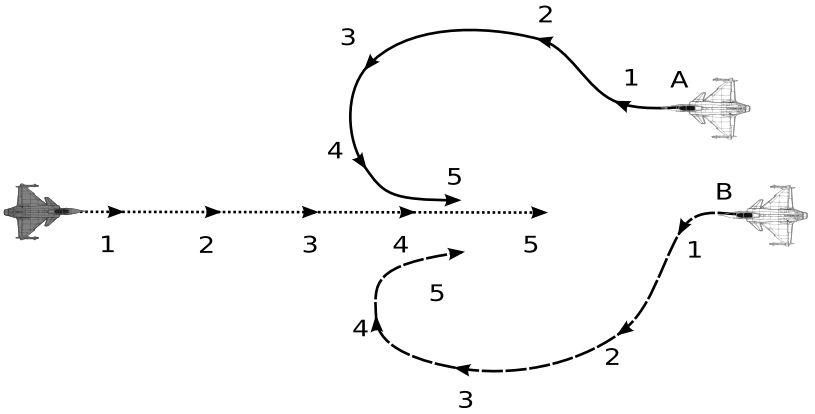
Obrázek 3.9: Taktika Drag and Bag. Stíhač A je ohrožen šedým stíhačem. Manévruje tak aby šedý stíhač o něj neztratil zájem a zároveň on sám nebyl zbytečně ohrožen, zatímco stíhač B se přiblíží pozice pro střelbu.

sebou nechají rozestup ne menší než minimální dostřel jejich zbraní a ne větší než maximální a v této formaci zaútočí. Pokud první z nich mine, může zaútočit druhý. Pokud je první napaden, druhý mu může poskytnout účinnou podporu (viz Drag and Bag, 3.4.1). Možnosti obrany proti tomuto manévru jsou stejné jako proti ofsetovému přiblížení jednoho stíhače s tou obměnou, že pokud stíhač provede úhyb směrem k útočícím stíhačům, velmi pravděpodobně souboj prohraje. Schéma ofsetového přiblížení z stejné strany je na obrázku 3.10. Tento typ útoku je ofenzivně účinný a stíhači si mohou poskytnout dostatečnou podporu.

Druhou možností ofsetového přiblížení formace je, že každý stíhač se přiblíží po jiné ofsetové trati, jeden po levé a druhý po pravé. Bránící stíhač nemá na vybranou - pokud provede úhyb směrem k libovolnému z útočníků, nabídne druhému ideální podmínky pro střelbu. Pro útočníky má takovýto manévr svá rizika - rozdělením se ztratí možnost vzájemné podpory. Toto riziko je ale vyváženo velkou útočnou silou. Stíhači by ale měli již před útokem naplánovat kdy a jak se opět spojí. Schéma ofsetového přiblížení z různých stran je na obrázku 3.11.



Obrázek 3.10: Ofsetové přiblížení formace - způsob přiblížení z jedné strany



Obrázek 3.11: Ofsetové přiblížení formace - způsob přiblížení z různých stran

Kapitola 4

Implementace multiagentní simulace leteckých soubojů

V minulých kapitolách byly popsány způsoby, jakými se tvoří nová prostředí pro Jason agenty, a jak se tito agenti programují a základy letecké taktiky. V této kapitole bude popsán návrh konkrétní multiagentní simulace leteckých soubojů. Nejdříve bude popsáno jakým způsobem je modelována problematika. Pak bude ukázána architektura implementace v Javě, popsáno rozhraní poskytované agentům, předvedena implementace agentů, kteří se v tomto prostředí budou pohybovat.

4.1 Tvorba modelu leteckého prostředí

Problém jsem formuloval takto: Názorným způsobem přenést letecké prostředí na obrazovku monitoru tak, aby taktiky popsané v kapitole 3 měly své opodstatnění i ve vytvořeném modelu.

Rozhodl jsem se ignorovat několik vlastností reálného prostředí. Především je to spojitost času. Čas ve vytvořeném modelu plyne krokově a jeden krok modelu je simulován zhruba jako jedna vteřina reálného času. Stav prostředí se mění jen v jednotlivých krocích. Rozhodl jsem se také ignorovat neurčitost informací reálného prostředí. Informace, které stíhači ve vytvořeném prostředí dostávají, jsou vždy přesné. Případné rozšíření o chybovost informací je triviální ale model by dle mého názoru ztratil na názornosti.

Názornost Prvně se zaměřme na názornost. Intuitivně lze vytušit, že letecký souboj se neodehrává v dvojrozměrném prostředí. Letadla mohou mít různou výšku, mohou klesat či stoupat. Mohou být různě nakloněná, mohou dokonce letět jinam než směřuje špička letadla. Takto komplexní prostředí je špatně uchopitelné i při 3D animaci, piloti proto při popisu manévrů nejčastěji používají pro předvedení různých poloh letadla ruce . Všechny manévry ale přesto jdou promítnout do jedné roviny, jak je ostatně vidět na předchozích ilustracích. Ztrácí se tím mnoho informací, které je třeba vysvětlit dodatečně, ale nejdůležitější parametry manévru zůstávají názorně vysvětleny. Pro tuto názornost jsem se naše prostředí rozhodl omezit na prostředí dvojrozměrné.

Možnosti letadel Z problematiky jsem vybral fakta, která mají nejzásadnější dopad na tvorbu leteckého modelu. Mezi tato fakta patří:

- Letadlo nemůže zastavit na místě, existuje tedy určitá minimální rychlost

- Pro letadlo taktéž existuje určitá rychlost maximální
- Letadlo nemůže skokově měnit svoji rychlost
- Letadlo nemůže skokově měnit svůj letový kurz
- Letadlo může měnit zároveň svoji rychlost i letový kurz

Tyto vlastnosti jsou modelovány takto:

- Model letadla má atribut rychlost - tento nejde změnit pod minimální rychlost R_{min} a nad maximální rychlost R_{max}.
- V rozmezí daném minimální a maximální rychlostí může změnit letadlo svoji rychlost v jednom kroku simulace maximálně o určité Δ_r.
- Letadlo může měnit svůj letový kurz v jednom kroku simulace maximálně o určité Δ_k

Dokud v systému proti sobě bojují pouze stejné typy letadel, na přesných hodnotách Δ_r a Δ_k příliš nezáleží. Všichni účastníci souboje mají stejné podmínky. Experimentálně jsme určili hodnoty $R_{min} = 100m/s$, $R_{max} = 200m/s$, $\Delta_r = 5m/s$ a $\Delta_k = 5deg/s$ jako vizuálně nejbližší diagramům z kapitoly 3, které jsem si dal za cíl napodobit.

Zbraňové systémy

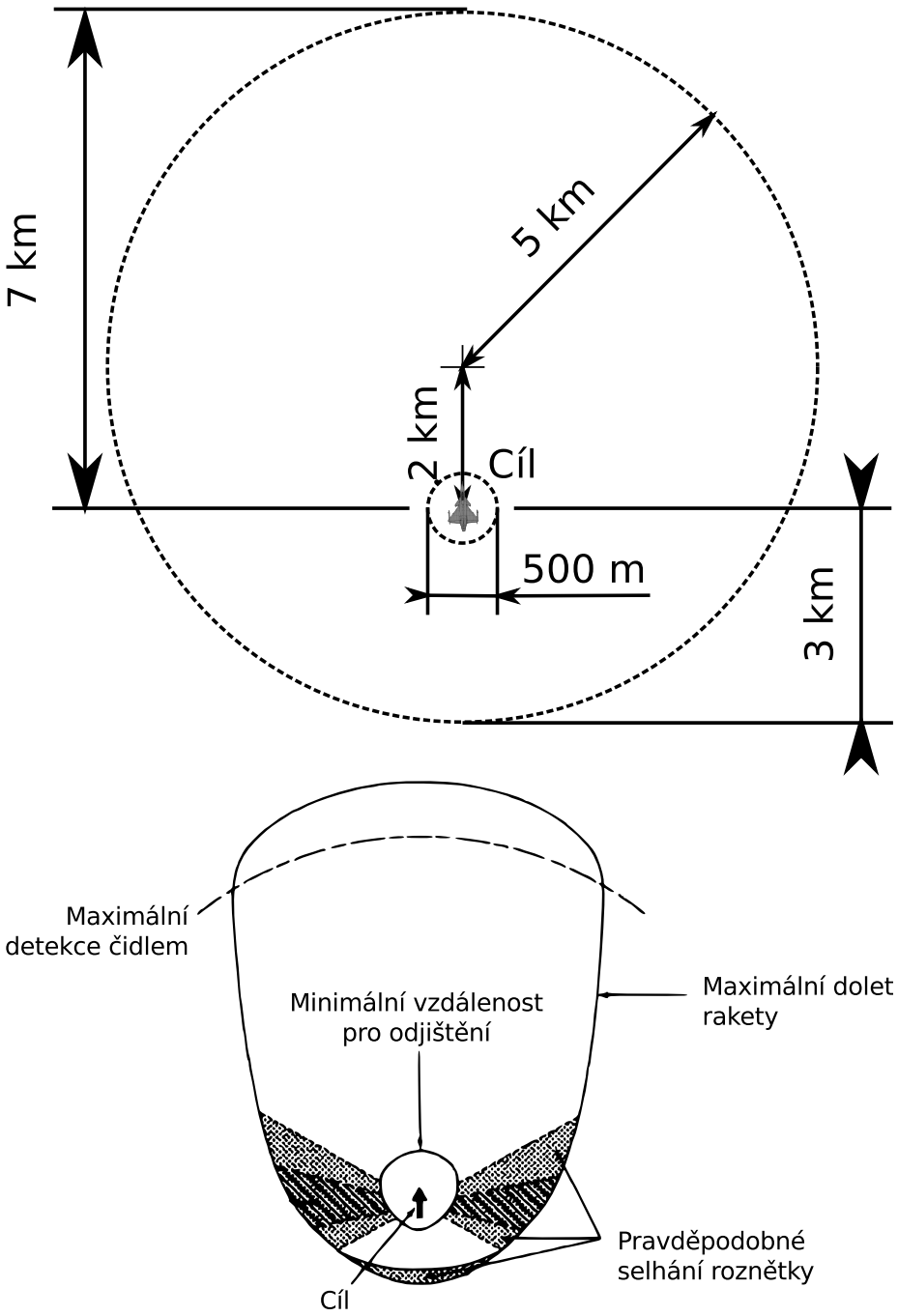
V prostředí jsou modelovány naváděné rakety. Jako vzor jsme si vybral nejběžněji používanou raketu AIM-9 Sidewinder. U raket jsem se rozhodl modelovat tyto vlastnosti:

- Doba letu rakety
- Minimální vzdálenost pro odpálení rakety
- Maximální vzdálenost určená doletem
- Maximální aspekt, ve kterém je raketa schopna cíl detekovat
- Výhodnost útoku ze zadní polosféry cíle a úspěšnost zásahu
- Omezený počet raket nesený jedním stíhačem

Tyto vlastnosti jsou modelovány následovně:

Doba letu rakety Dobu letu rakety jsem stanovil staticky na 10 kroků prostředí. Stíhač nemá příliš mnoho času na reakci, ale pokud hrozbu útoku zachytí včas, může se za dobu letu rakety otočit o již významných 50 stupňů. Raketa je vytvořena v okamžiku odpalu a po uplynutí 10ti kroků je vyhodnocen její dopad. V mezičase raketa v systému „neexistuje“.

Minimální a maximální dostřel U tohoto požadavku je třeba dosáhnout podobného tvaru pozic vhodných pro střelbu jako na obrázku 3.2. Minimální vzdálenost pro vypuštění rakety jsme určili 500 metrů, maximální na 5 kilometrů. Tyto hodnoty jsou založeny na příručce pro námořní letectvo [1]. Při střelbě zezadu je maximální dostřel zkrácen o vzdálenost, kterou cíl uletí za dobu letu rakety. S dobou letu rakety 10 kroků a maximální rychlostí nepřítele 200m/krok jsou výsledkem parametry pro střelbu znázorněné na obrázku 4.1. Podobnost není dokonalá, základní charakteristiky ale zachovává. Tohoto tvaru je v modelu dosaženo zapamatováním místa, kde byla vystřelena raketa, ale k vyhodnocení zda je cíl v dosahu dojde až po uplynutí doby letu.



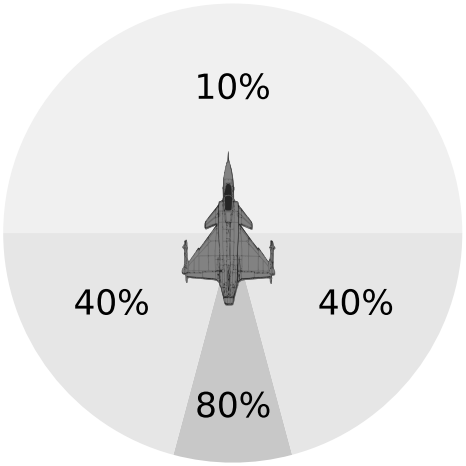
Obrázek 4.1: Nahoře: Maximální dostřel rakety s dostřelem 5km letící 10kroků při střelbě na nemanévrující cíl letící rychlostí 200m/krok.
Dole pro srovnání: Maximální dostřel rakety tak jak ji uvádí [11]

Maximální aspekt Modelovaná raketa Sidewinder je schopna se zaměřit zhruba na cíl v rozmezí 60°. Pokud je cíl mimo tento limit, raketa je neškodná. V modelu je tento fakt simulován tak, že raketa vždy mine pokud v okamžiku vystřelení je cíl mimo aspekt -30°až 30°.

Výhodnost útoku ze zadní polosféry Lerner v [6] uvádí pravděpodobnost zásahu při ideálních podmínkách až 80% a ideální pozici maximálně 30°od ocasu cíle. Mimo ideální podmínky je pravděpodobnost zásahu výrazně nižší, cca 10%. Lze předpokládat, že se pravděpodobnost zásahu snižuje s aspektem od nepřítele. Proto jsem zjednodušeně namodeloval pravděpodobnost zásahu jak je znázorněno na obrázku 4.2.

Omezený počet raket Běžně používaná letadla [1] nemohou vzlétnout s více jak 8mi raketami, většinou méně. Z toho důvodu jsem maximální počet raket nesený stíhačem stanovil na 8. Po vyčerpání raket je stíhač zcela bezbranný.

Aspekt od cíle	Pravděpodobnost zásahu
165 – 195	80%
90-165, 195-270	40%
jinak	10%



Obrázek 4.2: Pravděpodobnost zásahu cíle při střelbě z různých úhlů.

4.2 Prostředí

Prostředí je implementováno jako několik modelů. Je implementován model pro prostředí (`AirspaceModel`), ve kterém se pohybují modely letadel (`AircraftModel`). Pro agenty zprostředkovává prostředí třída `AirspaceEnv` a uživateli se souboje vizualizují prostřednictvím `AirspaceGUI`. Vizualizace používá knihovnu SWING a data pro zobrazení jsou aktualizována při každé změně modelu.

[1] Saab JAS-39 Gripen, Grumman F-14 Tomcat, Boeing F/A-18E/F Super Hornet, Sukhoi Su-33, Mikoyan MiG-29M, Sukhoi Su-27, Lockheed Martin F-35 Lightning II

4.2.1 Hlavní smyčka

Při návrhu prostředí je třeba určit jakým způsobem bude plynout čas v modelu. Je možno zvolit několik přístupů:

Krokované prostředí Prostředí čeká na agenty, všichni agenti musí ohlásit svou akci než prostředí učiní další krok. Všichni agenti mají tedy stejné podmínky a vylučují se externí vlivy na simulaci. Agenti mohou provádět libovolně složité výpočty, protože prostředí na ně počká. Je vhodné při simulaci prostředí, na které nemá vliv nic jiného než agenti v něm obsažení. Toto prostředí je již implementováno v systému Jason jako `TimeSteppedEnvironment`.

Nezávisle běžící prostředí Prostředí se pohybuje nezávisle na agentech. Agenti jsou průběžně informováni o stavu prostředí, i když sami neprovádějí žádné akce. Tento přístup je vhodný pro hry a animace, kde je nutno pravidelně zobrazovat snímky. Tento přístup není efektivní pokud je mezi událostmi v prostředí velký časový odstup a stav prostředí by se počítal zbytečně beze změn.

Tyto přístupy analogicky odpovídají *next-event* a *activity-scanning* technikám simulace.

V modelovaném prostředí dochází ke změnám i když agenti sami nevykonávají žádnou činnost (agenti se dokonce v prostředí neustále pohybují aniž by provedli libovolnou akci). Zároveň je nutno prostředí vizualizovat uživateli, tedy je zde požadavek animace. Proto jsem zvolil přístup nezávisle běžícího prostředí.

Konkrétní implementací je vytvořená třída `AirspaceEnv`. Tato třída poskytuje virtuální rozhraní agentům. Krokování prostředí je tvořeno smyčkou běžící v samostatném vlákně. Princip hlavní smyčky je ukázán v ukázce programu 1.

```
1: tick = 0;
2: while not end do
3:     prepocti model
4:     aktualizuj vjemy agentu
5:     upozorni agenty na zmenu vjemu
6:     odstran mrtve agenty
7:     aktualizuj GUI
8:     tick++
9:     cekej(cas jednoho kroku)
10: end while
```

Program 1: Hlavní smyčka programu

4.2.2 Model

Logiku domény jsem rozdělil na několik částí. Hlavní třídou je `AirspaceModel`, který se stará o správu letového prostoru a všech objektů v něm. Jeho odpovědností je udržovat přehled všech objektů v prostoru, vztahů mezi nimi a provádět výpočet kroků simulace. V prostoru se mohou vyskytovat letadla (třída `AircraftModel`) a rakety (třída `Missile`).

Model letadla se stará především o správu kurzu a rychlosti. Obsahuje vždy požadovanou hodnotu (kterou si vyžádal agent) a hodnotu aktuální. V každém kroku pak přiblíží aktuální hodnotu k té požadované o maximální možnou změnu příslušného parametru. V modelu

letadla je také uchovávána aktuální pozice letadla, všechny předchozí pozice letadla[2] a počet zbývajících raket.

Zodpovědností modelu rakety je výpočet pravděpodobnosti zásahu a následné určení zda raketa zasáhla. K tomuto účelu si pamatuje kde byla raketa vystřelena, kým a na koho.

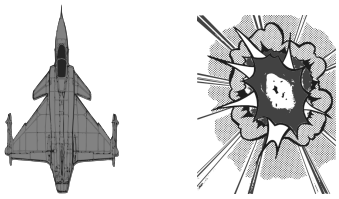
4.2.3 Pohled

Spuštěný program má jednoduché grafické rozhraní, které se snaží vizuálně replikovat diagramy z kapitoly 3. Znázornění se skládá ze 3 částí:

Aktuální pozice je znázorněna ikonou letadla. Její natočení odpovídá letovému kurzu. Pokud je letadlo sestřeleno, promění se ikona v znázornění výbuchu. Ikony jsou znázorněny na obrázku 4.3.

Předchozí trasa je znázorněna křivkou procházející všemi body, skrz které agent od počátku simulace proletěl.

Časová razítka se zobrazují podél předchozí trasy. Razítko se vykresluje každý desátý krok prostředí. Tak lze po skončení simulace lépe pochopit jak souboj probíhal.



(a) Ikona letícího letadla

(b) Ikona sestřeleného letadla

Obrázek 4.3: Ikona letícího letadla a ikona sestřeleného letadla

Programovací jazyk Java poskytuje vhodné nástroje pro přímé propojení modelu a pohledu. V hlavní smyčce programu (program 1) je zavolána metoda aktualizace a pohled data získá z modelu a aktualizuje svoje komponenty.

4.2.4 Řadič

Řadič má dvě základní funkce:

1. Poskytovat agentům informace na základě stavu modelu.
2. Přijímat od agentů požadavky na provedení akcí a převádět je na akce modelu.

Všechny aktualizace stavu agentů v Jasonu probíhají pomocí metody `addPercept`. Jedním voláním metody `addPercept` lze upravit pouze jeden vjem agenta. Úpravu všech vědomostí agentů jsem tedy shrnl ul pod jednu metodu `updateAgsPercept`, která postupně upraví

[2]Pokud by simulace běžela dostatečně dlouho, její paměťové nároky by díky uchovávání všech minulých pozic mohly přesáhnout dostupnou paměť. Při testování toto nebyl problém, simulace má jasný výsledek do 300 kroků. Kompletní historie se uchovává především kvůli zpětné analýze průběhu simulace a opravení tohoto problému je triviální, například omezením počtu uchovávaných kroků

všechny vjemy všech agentů. Tato metoda odpovídá řádku 4 v programu 1. Jakmile jsou vjemy upraveny, agentům je změna oznámena pomocí `informAgsEnvironmentChanged`. Tuto metodu poskytuje přímo výchozí Jason prostředí. V tabulce 4.1 je seznam vjemů, které jsou agentům každý krok prostředí obnovovány. Při vytvoření prostředí je každému agentovi navíc vloženo několik vjemů, které se po dobu běhu prostředí nemění.

Požadavky na provedení akcí v prostředí lze odchytávat dvěma způsoby. Každý požadavek o provedení akce je předán metodě `executeAction` i s jeho parametry. Pro prostředí s malým množstvím akcí lze v této metodě provést kompletní obsluhu. Pokud je třeba, je možno každou akci převést na samostatnou třídu v Javě, kde je možno akci obsloužit přehledněji. Jelikož všechny požadavky jsou pouze překládány na příslušnou metodu v modelu, který se stará o obsluhu, zvolil jsem první způsob. Všechny akce dostupné agentům jsou vyjmenovány v tabulce 4.2. Agent může požadovat akce kdykoliv, ale akce se provede až s dalším krokem programu. Pokud agent požaduje během jednoho kroku více akcí (například „zatoč vpravo“ a zároveň „zatoč vlevo“, provede se ta, která byla požadována později.

Funkci řadiče v implementovaném prostředí plní třída `AirspaceEnv`, dědící z třídy `Environment`, která je součástí Jasonu. Při inicializaci její instance se spustí hlavní smyčka programu.

4.3 Modelování taktik

K programování agentů v tomto prostředí je možno přistoupit z několik úhlů. Hlavní možné přístupy bych shrnul takto:

Plánující agent Plánující agent si vytvoří podrobný plán. Odhadne trasu nepřítele a naplánuje si dopředu vlastní trasu, kterou poletí. Výhodou tohoto druhu agenta je možnost velmi komplikovaných plánů a precizního provedení. Bohužel, takto plánující agent je neflexibilní. V prakticky neustále se měnícím prostředí (kterým letecké prostředí zajisté je), a vzhledem k potencionální komplexnosti výpočtů je téměř nemožné spočítat výsledky včas a zároveň přesně. Obzvlášť v souboji proti agentovi, který tak říkajíc nespolupracuje by bylo nutné znovu plánovat manévr po každém kroku prostředí.

Reaktivní agent Reaktivní agent na základě jemu přidělených pravidel reaguje na prostředí, ve kterém se nalézá. V každém kroku provádí znovu jednoduché vyhodnocení situace a dle jemu naprogramovaných plánů na ni reaguje. Jeho výhodou je nízká výpočetní náročnost. Jeho rozhodnutí mohou být nepřesná a on sám s tím počítá. Pokud dojde k rapidní změně situace, okamžitě na ni reaguje, protože není zatížen plány do budoucna.

Mezi těmito přístupy je třeba nalézt určitý kompromis a vyhovující aktuálnímu problému. V prostředí, které se nemění jinak než na základě agentových akcí, a ve kterém nezáleží příliš na včastnosti provedení akcí je jistě žádoucí agentovy akce naplánovat do podrobností. Je tak možné dosáhnout výrazně větší efektivity agenta. Naopak v prostředí, které se mění i bez přičinění agenta, a ve kterém hrozí agentovi riziko pokud nebude včasně jednat je potřebné reagovat ihned. Zdlouhavý výpočet nemusí být žádoucím, protože v okamžiku kdy je dokončen jsou jeho výsledky neaktuální.

Již na začátku práce jsem se rozhodl vytvořit agenty převážně reaktivní. Agenti jsou navrženi tak, že mají mnoho plánů pro obecné situace, mají recepty jak řešit základní problémy jako je např. změny ofsetu na požadovaný, změny vzájemného kurzu a tak dále. Jednotlivé manévry pak skládají z posloupnosti jednotlivých pravidel. Plánování agenta

Vjem	Význam
tick(Counter)	Počítač kroku prostředí. Je navýšen v po každém kroku
location(X, Y)	Poloha agenta v souřadnicovém systému
location(Agent, X, Y)	Poloha jiného agenta
heading(Heading)	Letový kurz agenta ve stupních
speed(Speed)	Současná rychlost agenta
heading(Agent, Heading)	Současný kurz jiného agenta
heading(Agent, Heading)	Současná rychlost jiného agenta
heading_to(Agent, Heading)	Kurz k jinému agentovi
heading_from(Agent, Heading)	Kurz od jiného agenta
aspect_to(Agent, Aspect)	Aspekt k jinému agentovi
reverse_aspect(Agent, Aspect)	Aspekt od jiného agenta
team(TeamNumber)	Tým, ke kterému agent náleží
team(Agent, TeamNumber)	Tým, ke kterému náleží jiný agent
distance(Agent, Distance)	Vzdálenost k jinému agentovi
angle_off_tail(Agent, AOT)	Vzájemný kurz k jinému agentovi
offset(Agent, Offset)	Ofset od jiného agenta
turn_radius(Radius)	Aktuální minimální poloměr zatáčky agenta
missiles(NumberOfMissiles)	Počet zbývajících raket

Tabulka 4.1: Seznam všech pravidelně atualizovaných vjemů agenta a jejich význam

Akce	Význam
heading(NewHeading)	Změna kurzu. na přesnou hodnotu.
speed(NewSpeed)	Změna rychlosti. NewSpeed může být číslo nebo řetězce `"up"`, `"down"`.
turn(Direction)	Relativní změna kurzu. Direction mohou být řetězce `"left"`, `"right"`.
fire(Target)	Vystřelení naváděné rakety na cíl. Odebere jednu raketu ze zbývajícího počtu raket. Pokud nezbývají žádné rakety, selže.

Tabulka 4.2: Seznam všech akcí dostupných agentům a jejich význam

je uskutečněno postupným přechodem mezi mentálními stavy. Stav agenta ovlivňuje, jak bude reagovat v různých situacích. Mezi stavy agent volně přechází na základě stanovených podmínek. V následujících jsou popsány jednotlivé taktiky a návrh jejich převodu do modelu. Konkrétní implementace jsou pak popsány v kapitole 4.4.

4.3.1 Přímé pronásledování

Přímé pronásledování bylo představeno v kapitole 3.3.2

Rozbor

Útočník se neustále snaží udržet aspekt na 0. Tedy letový kurz útočníka je shodný jako kurz k cíli. Je třeba znát pouze kurz k cíli. Pro vstup do přímého pronásledování nejsou žádné podmínky, je vždy vhodné. Pro opuštění přímého pronásledování nejsou žádné podmínky.

Implementace

Kurz k cíli získá agent z (`heading_to_enemy(EnemyAgent, Heading)`. Aby nedocházelo k různým substitucím `EnemyAgent`, je třeba si pamatovat na jaký cíl agent útočí. Ukázka provedení přímého pronásledování je na obrázku

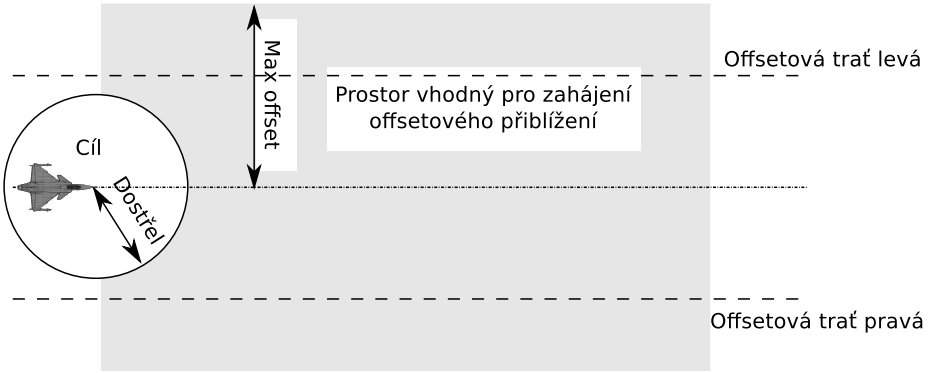
4.3.2 Ofsetové přiblížení

Ofsetové přiblížení bylo představeno v kapitole 3.3.3.

Rozbor

Ofsetové přiblížení se dá rozdělit do několika fází. Tyto fáze jsou:

Vyhodnocení vhodnosti útoku Útok je z přední polosféry cíle. Útok je tedy nevhodný pokud se nacházíme mimo přední polosféru. Útok je nevhodný pokud jsme k cíli příliš blízko. Minimální vzdálenost je závislá od dostřelu protivníka. Není žádoucí se otáčet k obránci bokem v okamžiku kdy na útočníka může obránce střílet. Útok je také nevhodný při příliš velkém ofsetu. V takovém případě by se měl útočník spíše pokoušet k cíli přiblížit. a případně po snížení ofsetu přejít do ofsetového přiblížení. Prostor vhodný pro zahájení ofsetového útoku je znázorněn na obrázku 4.4.



Obrázek 4.4: Vhodné pozice pro zahájení ofsetového útoku (šedě).

Získání ofsetu Je třeba zvolit optimální ofset. [1] uvádí ideální separaci 20 000 stop, tedy cca 6km, tuto hodnotu jsme převzali. Pro nejrychlejší získání ofsetu by se měl útočník otočit kolmo k pomyslné ose cíle a letět tímto směrem dokud není získán dostatečný ofset. Volbu ofsetové tratě jsem zjednodušil na levou, pokud je útočník od cíle vlevo a pravou pokud je útočník od cíle vpravo. Rozhodnutí, zda je třeba pro získání ofsetové tratě zatočit vlevo nebo vpravo záleží na vzájemné pozici a natočení agentů.

Udržení ofsetu a přiblížení Již před získání ofsetové tratě je třeba začít zatáčet směrem k cíli. Kdy je třeba zatáčet se odvíjí od poloměru zatáčky stíhače, zatáčet tedy stíhač musí začít o poloměr zatáčky dříve než dosáhne ofsetové tratě. Po získání optimálního ofsetu udržuje stíhač kurz přesně opačný než má protivník, vzájemný kurz 180°.

Přechod do útoku Po zkrácení vzdálenosti přejde stíhač do konečného útoku. Vhodná vzdálenost pro přechod je diktována možnou úhlovou rychlostí zatáčky útočníka a vzájemnou vzdáleností obránce a útočníka. Dle [11], [1] by tento přechod měl být prováděn tak, aby nos útočníka celou dobu směřoval na cíl. Tím se sníží možnost detekce útočníka obráncem. Z toho vyplývá, že je třeba si pamatovat změnu aspektu k cíli a pokud se velikost této změny začne blížit maximální úhlové rychlosti útočníka (5°), začít zatáčet k cíli. Další možností je přejít do útoku pokud se cíl příliš přiblíží k útočníkovi. Tuto druhou podmínku jsem stanovil na 7km, tedy na dostřel rakety při míjení agentů zepředu.

Implementace

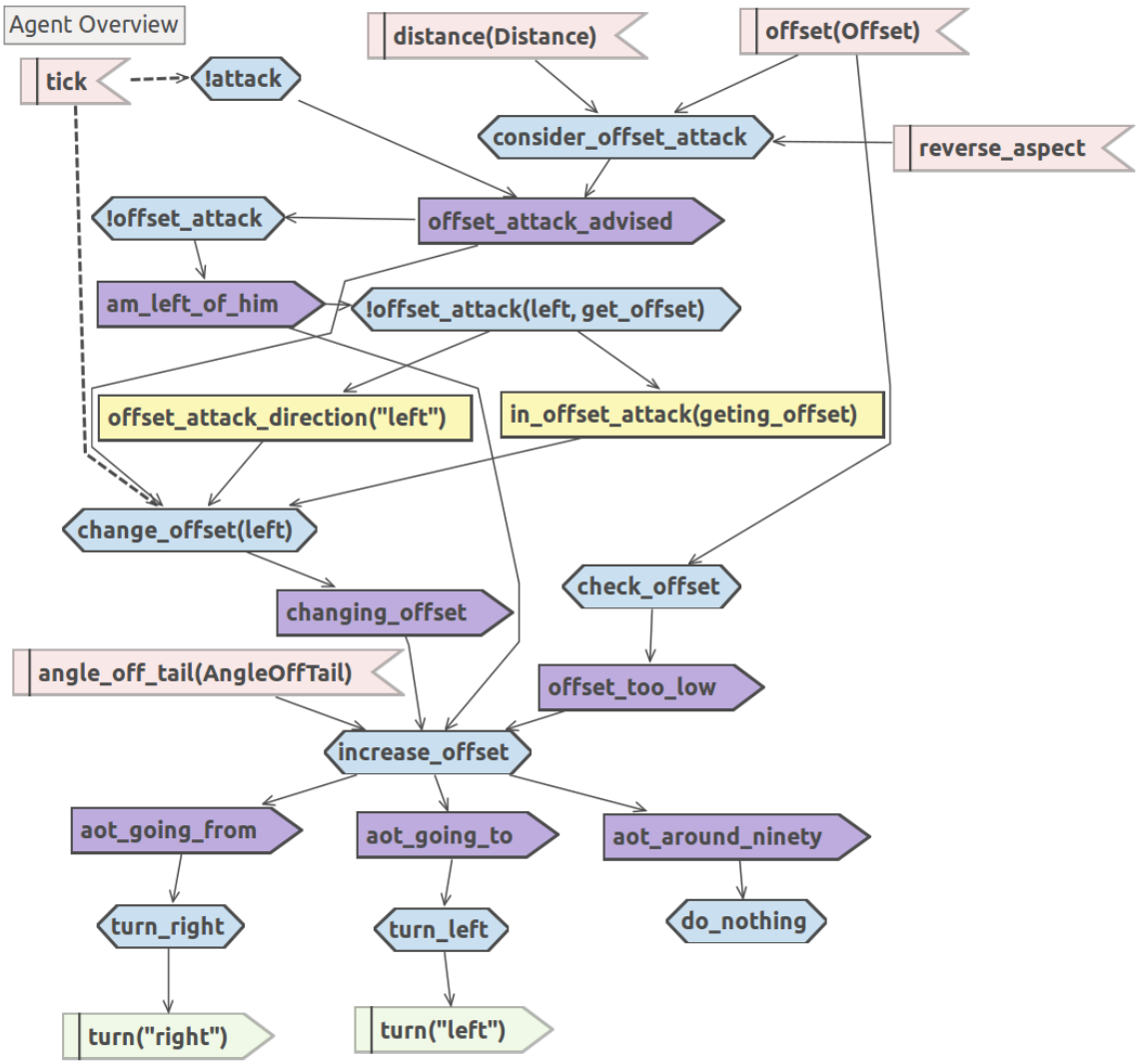
Jednotlivé fáze jsou implementovány jako jednotlivé stavy mezi kterými agent přechází.

Vyhodnocení vhodnosti útoku Ofsetový útok je vhodný pokud kombinace následujících faktů:

- Vzdálenost (distance(Agent, Distance)) je vyšší než dostřel (rocket_range(RocketRange))
- Aspekt od cíle k nepříteli (aspect_from(Agent, Aspect))je nižší než 90 (tedy cíl směřuje k útočníkovi)
- Ofset (offset(agent, Offset)) je nižší než definovaný maximální ofset (7km)

Získání ofsetu Agent se rozhodne zda chce dosáhnout levé nebo pravé ofsetové tratě. Pokud je vlevo od cíle, vybere levý ofset, pokud vpravo, pravý. Následně se natočí tak aby vzájemný kurz byl 90°pokud je vpravo a -90°pokud je vlevo od cíle. Fáze je ukončena jakmile ofset je vyšší než požadovaný ofset +- poloměr zatáčky nebo ofsetový útok a agent tím přejde do fáze udržení ofsetu. Tento proces je poměrně komplikovaný, náhled do něj je na obrázku 4.5.

Udržení ofsetu a přiblížení Agent provede zatáčku tak, aby vzájemný kurz k cíli byl 180°. V každém kroku si pamatuje změnu aspektu od cíle k němu. Přibližováním se k cíli se změna aspektu od cíle zvyšuje. Fáze je ukončena jakmile se změna aspektu přiblíží maximální úhlové rychlosti zatáčky agenta, nebo se agent dostane na dostřel k cíli. Další možností ukončení fáze přiblížení je pokud cíl proletí kolem agent (tedy agent ho vidí na aspektu 90°. Agent pak přejde do fáze koncový útok.



Obrázek 4.5: Rozhodovací proces zda agent pro získání ofsetu zatočí vlevo či vpravo. Celý proces začíná přidáním vjemu tick. Volá se plán *attack*, který pokud je ofsetový útok vhodný, předá řízení plánu *offset_attack*. Ten rozhodne zda má zaujmout agent ofsetovou trat levou nebo pravou na základě informace zda je agent vlevo či vpravo. Při dalším kroku prostředí se již agent snaží zaujmout ofsetovou trať. Nejdříve zjistí zda je jeho ofset moc nízký nebo moc vysoký (*offset_too_low*) a v tomto případě se rozhodne že ofset je pro zaujetí ofsetové tratě nutno zvýšit. Na základě vzájemného kurzu se rozhodne, zda je třeba pro zaujetí ideální trasy pro zvýšení ofsetu zatočit doleva nebo doprava. Pokud je vzájemný kurz přibližně 90, pokračuje v stávajícím kurzu - rychleji ofset zvyšovat nemůže.
V diagramu je pro přehlednost vyjádřena jen jedna cesta, kterou se rozhodování ubírá. Je v něm předpokládáno, že útočník je vlevo od cíle, ofsetový útok je vhodný a ofset je příliš nízký.

Koncový útok Agent ukončí ofsetový útok a přejde do přímého pronásledování.

Pokud v libovolné fázi ofsetového útoku dojde k tomu, že by útok přestal být vhodný, agent ukončí ofsetový útok a přejde do přímého pronásledování.

4.3.3 Ofsetové přiblížení v týmu

Ofsetové přiblížení v týmu bylo představeno v kapitole 3.4.2. Pro implementaci této taktiky je třeba navrhnout způsob komunikace mezi agenty.

Rozbor

Pro použití ofsetového přiblížení v týmu je třeba aby byly naplněny tyto podmínky:

- Existují agenti v jeho týmu, kteří jsou schopni provést ofsetové přiblížení v týmu
- Ofsetové přiblížení musí být vhodné.

Ofsetové přiblížení v týmu je tedy nástavbou nad ofsetovým přiblížením. Tým agentů musí pro ofsetové přiblížení v týmu provést tyto kroky:

- Vybrat vedoucího
- Vedoucí zvolí cíl útoku
- Vedoucí rozdělí ofsetové tratě (levá/pravá) mezi jeho podřízené
- Agenti provedou ofsetový útok po přidělené ofsetové trati na přidělený cíl

Implementace

Výběr vedoucího Využil jsem zde toho, že agenti jsou číslovaní (např. plane0, plane1 atd) a agent zná svoje pořadové číslo. Pomocí faktu `team(Agent, TeamNumber)` zjistí pořadová čísla všech ostatních agentů z týmu a uloží je do seznamu. Po setřídění seznamu je první agent seznamu zvolen vedoucím. Tato metoda závisí na přesném pojmenování agentů a na tom, že prostředí pravidelně odstraňuje fakta o mrtvých agentech.

Výběr cíle Výběr cíle jsem se rozhodl zjednodušit. Pilot ve skutečném prostředí by se rozhodoval na základě vzdálenosti, pozice a mnoha dalších parametrů. V našem prostředí, kde agenti jednoho týmu startují pohromadě, a není tedy mezi agenty jednoho týmu výrazný rozdíl, může vedoucí vybrat cíl formace náhodně. Výběr cíle provede podle tohoto pravidla:

```
enemy(Agent) :-
        team(Agent, OtherTeam) &
        team(MyTeam) &
        MyTeam \== OtherTeam.
```

Kód 4.1: Ukázka výběru nepřítele k napadnutí

Vybere prvního agenta, který zároveň není v jeho týmu. Z experimentálních výsledků ale vyplývá, že je to většinou agent jiného týmu s nejnižším pořadovým číslem.

Výběr ofsetových tratí Vedoucí rozděluje levou a pravou ofsetovou trať střídavě svým podřízeným agentům.

Provedení ofsetového útoku Každý agent, který dostal „rozkaz“ k provedení ofsetového útoku, provede ofsetový útok. Při výběru ofsetové tratě se řídí tratí přidělenou vedoucím. Jinak agenti postupují samostatně dle taktiky Ofsetové přiblížení.

4.3.4 Komunikační protokol

Pro dosažení domluvy mezi agenty je třeba specifikovat alespoň základní komunikační protokol. V rámci implementované taktiky Ofsetové přiblížení v týmu je třeba, aby si agenti byli schopni sdělit:

Jaký cíl vedoucí vybral Vedoucí pošle všem podřízeným nově zaměřeného stíhače. Je třeba aby y se podřízení nějak zachovali k jejich současnému cíli, pokud nějaký mají. Vedoucí nechává vyřízení těchto podrobností na podřízených a posílá pouze rozkaz k změně cíle a nový cíl.

Jakou ofsetovou trať vedoucí přidělil podřízenému Vedoucí pošle všem podřízeným jejich přiřazenou ofsetovou trať. To, že vybraná ofsetová trať je vhodná, je na vedoucím. Vedoucí posílá pouze informaci o přidělené ofsetové trati.

4.4 Implementace taktik agenty

V této části práce bude předvedena ukázková implementace agentů v Jasonu, kteří budou schopni aplikovat strategie přímého pronásledování, ofsetového přiblížení a ofsetového přiblížení v týmu. Agenti budou schopni vyhodnotit kdy je která strategie vhodná a následně provést vhodnou strategii. Nakonec budou schopni rozpoznat vhodné podmínky pro střelbu a v rámci prostředí „sestřelit“ nepřítele.

4.4.1 Architektura agenta

Agent se skládá z hlavního souboru, do kterého jsou inkludovány jednotlivé knihovny s taktikami a schopnostmi. Jedna knihovna obsahuje jednu taktiku nebo dovednost (např. přímé pronásledování nebo střelba). V hlavním souboru agenta pak agent specifikuje jakým způsobem se mají jednotlivé knihovny využívat. Každá knihovna specifikuje pouze pravidla a plány jak lze splnit úkoly. Vložením knihovny do těla agenta by se nemělo změnit chování agenta (například tím, že by odchytávala všechny události nějakého typu) dokud tak agent výslovně neřekne.

4.4.2 Vyhodnocení vhodné taktiky

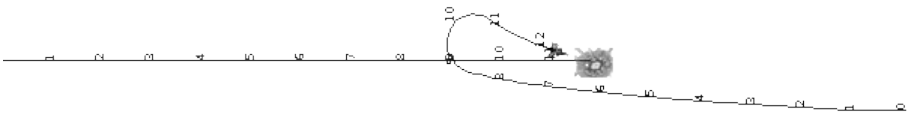
V mojí implementaci mají agenti určené uspořádání taktik dle pořadí, v jakém je žádoucí aby byly aplikovány. Toto pořadí je:

1. Týmové ofsetové přiblížení
2. Ofsetové přiblížení
3. Přímé pronásledování

Každá taktika má definované podmínky, při kterých je vhodná. Agent prochází taktiky od nejvíce žádoucích a zvolí k uskutečnění první taktiku, která je vhodná. Pokud taktika přestane být vhodná, agent provede znovu vyhodnocení vhodné taktiky.

4.4.3 Přímé pronásledování

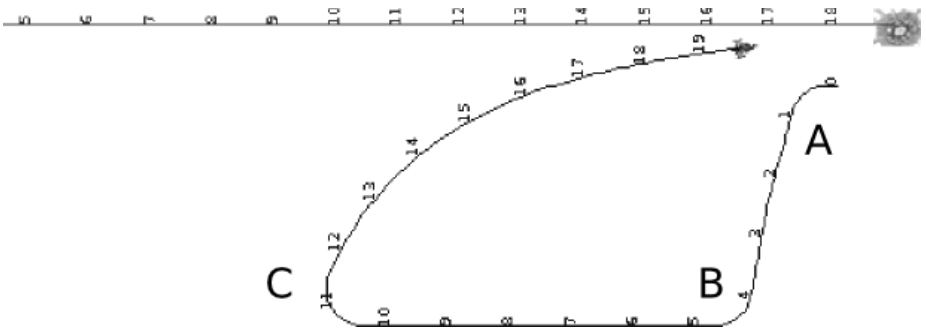
Při přímém pronásledování míří útočník přímo na obránce. Výsledek implementace je znázorněn na obrázku 4.6. Útočník celou dobu letu upravuje svůj kurz. Stíhači se pak míjejí, útočník otáčí a dostává se do ideální pozice pro střelbu.



Obrázek 4.6: Přímé pronásledování. Stíhač vpravo implementuje přímé pronásledování, stíhač vlevo nikoli. Stíhači se míjejí v čase 9, v čase 12 je stíhač začínající vlevo sestřelen.

4.4.4 Ofsetové přiblížení

Výsledek implementace ofsetového přiblížení je znázorněn na obrázku 4.7. Jsou jasně rozeznatelné jednotlivé fáze útoku, tak jak byly navrženy v kapitole 4.3.2. Na obrázku 4.8 je příklad, jak reaguje agent na změnu kurzu cíle. Na obrázku



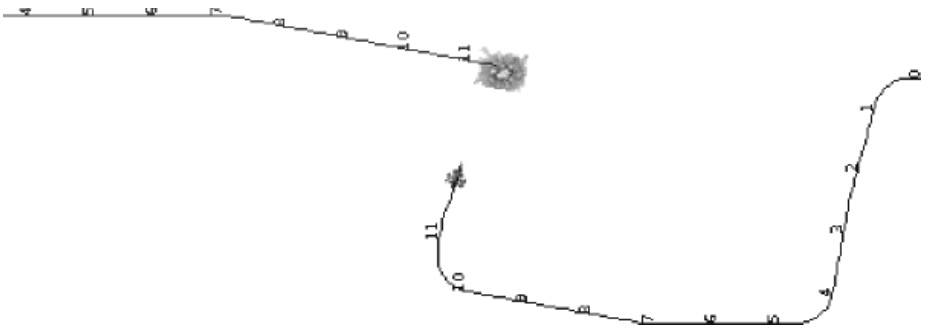
Obrázek 4.7: Ofsetové přiblížení. Útočník (začíná vpravo) začíná získávat ofset v bodě A, v bodě B začíná udržovat ofset, v bodě C obrací k cíli a pouští se do pronásledování. Nakonec dostihuje svůj cíl v čase 19

4.4.5 Ofsetové přiblížení v týmu

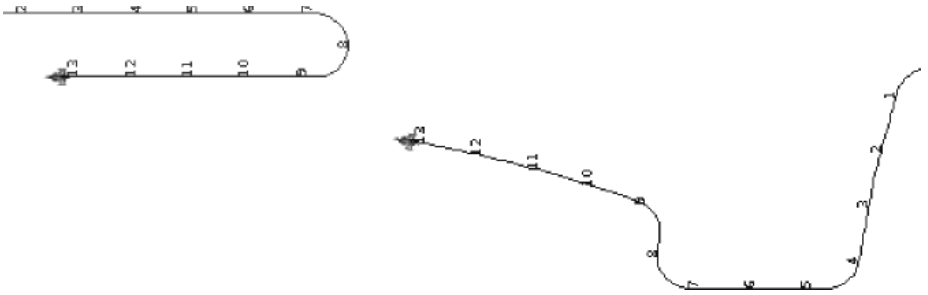
Výsledek implementace ofsetového přiblížení v týmu je znázorněn na obrázku 4.10.

4.5 Porovnání spolupracujících agentů s nespolupracujícími v simulovaném souboji

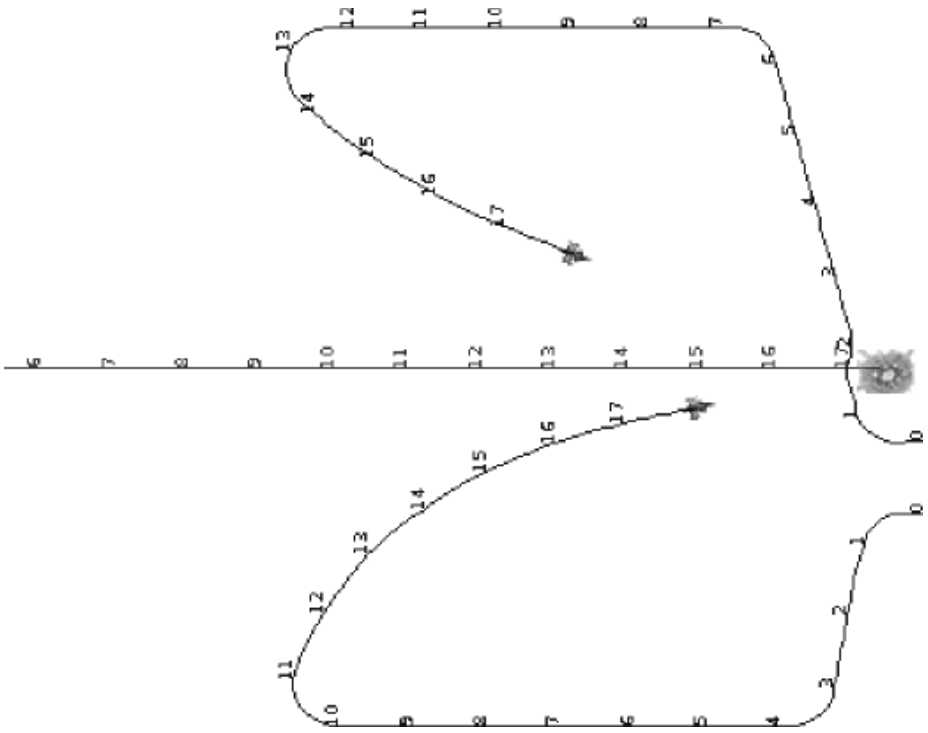
V této části budou popsány experimenty vedoucí k závěru, zda je pro agenty v simulovaném prostředí výhodnější spolupracovat. V rámci práce byli vytvořeno několik typů agentů.



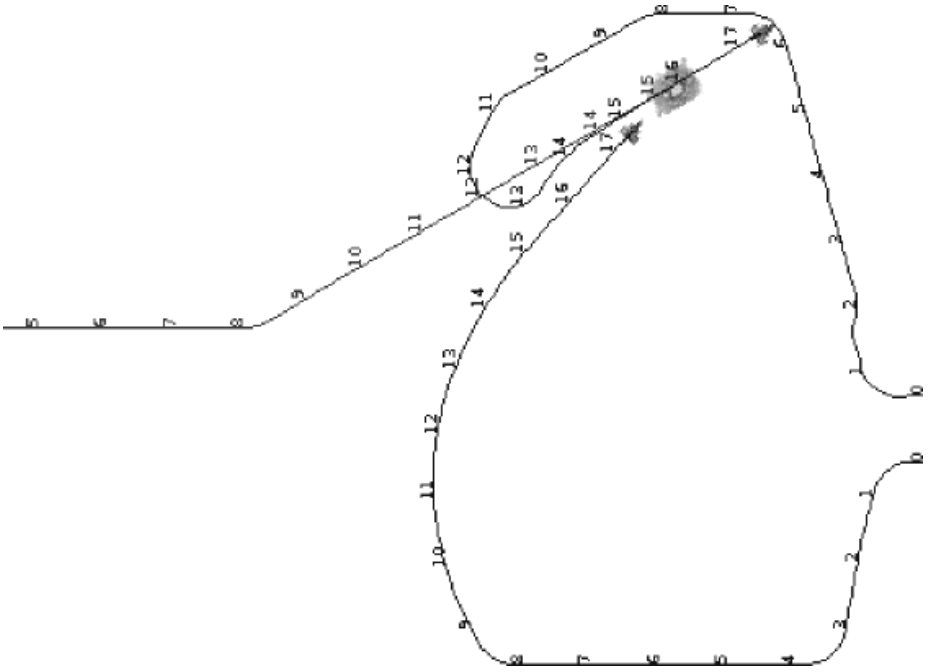
Obrázek 4.8: Reakce v ofsetovém přiblížení. Obránce (začíná vpravo) v čase 7 provede drobnou změnu směru. Útočník koriguje svůj manévr dle obránce.



Obrázek 4.9: Reakce v ofsetovém přiblížení. Obránce (začíná vpravo) v čase 7 otočí a pokusí se útočníkovi uniknout. Útočník se nejdříve snaží manévr korigovat jako na obrázku 4.8, jakmile zjistí, že pokračování v ofsetovém přiblížení nemá smysl, přechází na přímé pronásledování.



Obrázek 4.10: Ofsetové přiblížení v týmu.



Obrázek 4.11: Ofsetové přiblížení v týmu. V čase 8 provádí cíl úhyb vlevo. Stíhač začínající níže se tím dostává mimo parametry ofsetového útoku a přechází do přímého pronásledování. Stíhač začínající výše se nejdříve koriguje svůj kurz pro udržení vzájemného kurzu a v čase 11 přechází útoku.

1. Agent typu „cíl“. Neprovádí žádné manévry, umí pouze vystřelit pokud nějaký nepřítel splní parametry pro odpal rakety. Slouží pro testování ostatních. V kódu je označován jako `drone`.

2. Agent implementující přímé pronásledování. Dovede střílet po ostatních agentech a implementuje taktiku přímého pronásledování. Vybere si cíl a cíl drží dokud cíl není zničen. Nijak nespolupracuje s ostatními agenty v jeho týmu

3. Agent implementující ofsetové přiblížení. Je nástavbou agenta s přímým pronásledováním. Pokud je vhodné, použije ofsetové přiblížení. Nijak nespolupracuje s ostatními agenty v jeho týmu

4. Týmový agent. Je nástavbou agenta s ofsetovým přiblížením. Implementuje navíc taktiku ofsetového přiblížení v týmu.

Mezi různě pokročilými implementacemi agentů jsem následně uspořádal turnaj jeden proti jednomu, dva proti jednomu a dva proti dvěma.

Parametry souboje Souboj probíhal 300 kroků simulace a každý typ agenta získal tolik bodů, kolik zvládl zničit soupeřů. Týmy začínají proti sobě na vzdálenost 18km s offsetem 1km mezi jednotlivými letadly týmu. Každý souboj se opakoval 10krát s výměnou stran po třetím souboji. Po dosažení 300 kroků simulace byly souboje zastaveny a vyhodnoceny.

Výsledky soubojů Souboje vyhrávali pravidelně agenti zvládající ofsetové přiblížení. Jejich výhoda je hlavně vidět proti agentům typu cíl. Agenti v přímém pronásledování se

	Cíl	Přímé	Offset	Offset v týmu	Celkem
Cíl	1:1	1:9	0:10	0:10	**2:30**
Přímé	9:1	6:8	6:5	6:7	**27:21**
Offset	10:0	5:6	8:4	7:7	**30:17**
Offset v týmu	10:0	7:6	7:7	7:6	**31:19**

Tabulka 4.3: Výsledky soubojů jeden na jednoho. Udává poměr sestřelil:byl sestřelen dosažených druhem agenta za 10 soubojů. Je možné, aby součet sestřelů za 10 soubojů byl větší než 10, stíhači se mohou sestřelit navzájem.

k cíli blíží zepředu a zbytečně se tak vystavují střelbě. Jak již bylo uvedeno v kapitole 3, takovýto útok dává obránci i útočníkovi stejnou šanci na vítězství. Agenti zvládající ofsetový útok se tomuto nebezpečí vyhýbají a jejich výsledky jsou tedy lepší.

Souboje agentů s přímým pronásledováním a agentů zvládajících ofsetový útok probíhaly všechny podobně. Napadený ofsetový agent se snaží získat ofset, ale agenti s přímým pronásledování mu to neumožňují - stálým natáčením se na zacíleného agenta udržují na 0. Tím se dostávají do stálé zatáčky za agentem, který se snaží získat ofset. Nenapadený ofsetový agent tím má prostor získat ofset, přiblížit se k agentům s přímým pronásledováním a v rychlém sledu je sestřelit. U netýmových ofsetových agentů, kteří si pro útok oba vybrali stejnou stranu (viz 3.4.2) jde výrazně vidět vzájemná podpora agentů útočících s odstupem, ačkoli agenti spolu nijak nekomunikují.

U souboje agentů s přímým pronásledováním a agentů zvládajících ofsetové přiblížení v týmu jdou vidět výhody i nevýhody přiblížení z různých stran. Jeden z agentů musí bojovat v přečíslení, než se druhý dostane do pozice, ze které mu může pomoci. Přečíslený agent je pak často sestřelen dříve, než mu druhý agent z jeho týmu stihne pomoci. Tento problém by bylo vhodné vyřešit tak, že by agenti na sebe počkali a postupovali po svých ofsetových tratích s vzájemným aspektem cca 90 stupňů. Tím by se snížila separace mezi nimi a nenapadený agent by mohl rychleji zasáhnout.

Výsledky soubojů agentů zvládajících ofsetový útok a agentů zvládajících ofsetový útok v týmu se stávají zajímavými především v souboji dva na dva. Zde se projevuje zvýšená podpora mezi letadly v ofsetovém přiblížení ze stejné strany.

	Cíl	Přímé	Offset	Offset v týmu
2x Cíl	1:0	5:13	0:20	0:20
2x Přímé	10:2	10:3	10:7	10:7
2x Ofset	10:0	10:1	10:6	10:7
2x Ofset v týmu	10:0	10:0	10:8	10:7

	Celkem v přesile	Celkem v oslabení	Celkem
Cíl	6:53	2:31	8:84
Přímé	40:19	17:35	57:54
Offset	40:14	42:30	82:44
Offset v týmu	40:15	41:30	81:45

Tabulka 4.4: Výsledky soubojů dva na jednoho. V buňkách je uveden poměr sestřelil/byl sestřelen pro jednotlivé typy agentů. Celkem bylo provedeno 10 soubojů, s výměnou stran po 5ti soubojích.

	2x Cíl	**2x Přímé**	**2x Ofset**	**2x Ofset v týmu**	**Celkem**
2x Cíl	1:0	3:20	0:20	0:20	**4:60**
2x Přímé	20:3	13:13	0:20	0:20	**33:56**
2x Ofset	20:0	20:0	16:15	20:15	**76:30**
2x Ofset v týmu	20:0	20:0	15:20	18:13	**73:33**

Tabulka 4.5: Výsledky soubojů dva na dva. V buňkách je uveden poměr sestřelil : byl sestřelen pro jednotlivé typy agentů. Celkem bylo provedeno 10 soubojů, s výměnou stran po 5ti soubojích. Tabulka ukazuje jasnou převahu agentů zvládajících ofsetové přiblížení.

Kapitola 5

Závěr

Implementoval jsem prostředí pro multiagentní systém a několik vzorových agentů v tomto prostředí. Vytvořil jsem plány agentům pro vybrané letové manévry jednotlivých strojů a týmů. Výsledné modelované prostředí je záměrně velmi obecné a především se zaměřuje na poskytnutí co největšího množství informací agentům. Následné plánování akcí a odvozování nových skutečností je na agentech samotných. Výsledný agent je sice obsáhlejší, ale umožňuje širokou škálu chování. Nakonec jsem otestoval různé implementace v simulovaném souboji. Ukázalo se, že spolupracující agenti mají mírnou výhodu oproti agentům nespolupracujícím.

Největším překvapením v průběhu práce bylo, že simulace leteckých soubojů bez alespoň základní simulace zbraňových systému je bezpředmětná a vede k výsledkům, které zdaleka neodpovídají literatuře. Implementovaná simulace zbraňových systémů je minimální taková, že taktiky použitelné v reálném světě mají opodstatnění a souboje vyhrávají agenti chovající se dle literatury.

Prostředí, které bylo vytvořeno je zábavné pro práci a dovedu si představit jeho využití k výukovým účelům. Pro vytvoření základního chování je třeba pouze minimálních dovedností na straně programátora a sofistikovanost implementace je možno postupně rozšiřovat o komplexnější plány, komunikaci mezi agenty a tak dále. Další vývoj prostředí by se měl ubírat směrem k zavedení neurčitosti a zpoždění do informací dostupných agentům. Vedlo by to k možnosti použití širšího spektra taktik, založených na reakční době.

Jednotlivé druhy chování agentů jsou zjednodušené a je možné je dále rozvinout. Další vývoj agentů by se ubíral směrem k implementaci zpožděného a předsazeného pronásledování a větší sofistikovanosti výběru cíle.

Literatura

[1] *Flight Training Instruction For Air To Air Intercept Procedures Workbook*. Naval Air Training Command, 9 2010.
URL http://www.cnatra.navy.mil/pubs/folder5/NFO_SNFO/P-825.pdf

[2] Behrens, T. M.; Dix, J.; Hindriks, K. V.: Towards an Environment Interface Standard for Agent-Oriented Programming. *Ifl Technical Report Series*, 9 2009, ISSN 1860-8477.

[3] Bordini, R. H.; Hübner, J. F.; Wooldridge, M.: *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. Wiley-Interscience, první vydání, 11 2007, ISBN 9780470029008.
URL http://amazon.com/o/ASIN/0470029005/

[4] Bratman, M. E.: *Intention, Plans, and Practical Reason (David Hume Series)*. Center for the Study of Language and Information, 3 1999, ISBN 9781575861920.
URL http://amazon.com/o/ASIN/1575861925/

[5] Krasner, G. E.; Pope, S. T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.*, ročník 1, č. 3, Srpen 1988: s. 26–49, ISSN 0896-8438.
URL http://dl.acm.org/citation.cfm?id=50757.50759

[6] Lerner, P.: Sidewinder. *Air & Space magazine*, 11 2012.
URL http://www.airspacemag.com/military-aviation/Sidewinder.html

[7] Odell, J.: Objects and agents compared. *Journal of Object Technology*, ročník 1, 2002: s. 41–53.

[8] Padgham, L.; Winikoff, M.: *Developing Intelligent Agent Systems: A Practical Guide (Wiley Series in Agent Technology)*. Wiley, první vydání, 7 2004, ISBN 9780470861202.
URL http://amazon.com/o/ASIN/0470861207/

[9] Rao, A. S.: AgentSpeak(L): BDI Agents speak out in a logical computable language. 1996.

[10] Rao, A. S.; Georgeff, M. P.: BDI Agents: From Theory to Practice. 1995.

[11] Shaw, R. L.: *Fighter Combat: Tactics and Maneuvering*. Naval Institute Press, 6 vydání, 12 1985, ISBN 9780870210594.
URL http://amazon.com/o/ASIN/0870210599/

Příloha A

Manuál

Prostředí bylo implementováno pro Java JDK 1.7 a Jason 1.3.8 a testováno na platformách Windows 7 a Ubuntu Linux 12.10.

Spuštění práce

Pro spuštění práce je třeba mít instalovánu Java JDK ve verzi 1.7. Postup spuštění programu (převzato od Ing. Jana Samka, Ph.D.[1], aktualizováno a upraveno):

1. Spustit `Jason1.3.8\bin\jason.bat` - toto spustí Jason IDE
2. nastavit cestu Javu Home (menu *Plugins* → *Plugin Options* → *Jason* a nastavit cestu Java Home na: `C:\ProgramFiles\Java\jdk1.7.0_11\`)
3. Otevřít projekt (menu *File* → *Open* → `src\air_conditioner.mas2j`)
4. Spustit projekt (menu *Plugins* → *Jason* → *Run Project*)

Změna spouštěných agentů

Kteří agenti se mají spustit je specifikováno v souboru `src/air_conditioner.mas2j`.

```
1  MAS air_conditioner {
2          infrastructure: Centralised
3          environment: airspace.AirspaceEnv("air_conditioner.mas2j",2,1)
4
5          agents:
6                  plane0 offsetonly;
7                  plane1 offsetonly;
8                  plane2 drone;
9
10         aslSourcePath:
11                 "src/asl";
12 }
```

Je třeba dodržet názvy a pořadí číslování agentů (plane0, plane1 atd). Na řádku 3 se předává prostředí počet agentů a jejich rozdělení do týmů (v ukázce 2, 1 - první tým má dva členy,

[1] http://www.fit.vutbr.cz/~samejan/w/doku.php?id=vyuka:ags-2013:cv1

druhý jednoho, celkem 3 agenti). Agenti se do týmů přiřazuji v takovém pořadí v jakém jsou zapsaní v sekci `agents` (řádek 5).

Jiné agenty je možno spustit přepsáním sekce `agents`. Je možno použít libovolného agenta ze složky `src/src/asl`. Agenti se identifikují názvem souboru, bez přípony (například agent `src/src/asl/drone.asl` je v `air_conditioner.mas2j` identifikován jako `drone`). Nové agenty přidáte přidáním záznamu do sekce `agents`. Je třeba dát pozor na správné pojmenování agenta (`planeX`), ukončení řádku středníkem a navýšení počtu agentů v řádku 3. Na testovacích počítačích nečinilo problém prostředí s 16ti soupeřícími agenty v týmech po 8mi.